Computer & Literatur C&L

Michaël Van Canneyt

Free Pascal 2

Handbuch und Referenz

Win32/64, CE, OS X
und Linux/Unix

Das deutsche Originalhandbuch

Software:

Free Pascal ist ein moderner und plattformunabhängiger Compiler für die Programmiersprache Object Pascal. Er bildet auch die Unterlage für fortgeschrittene Entwicklungsumgebungen wie Lazarus und Morfik AppsBuilder.

Buch:

Zu Free Pascal gibt es im Internet eine mehrtausendseitige Dokumentation, die für diese autorisierte deutsche Fassung übersetzt und ediert wurde. Diese deutsche Version enthält in teilweise gestraffter und ergänzter Form das Benutzerhandbuch, die komplette Sprachbeschreibung, die Erläuterung aller Kommandozeilenschalter und Metabefehle sowie die Beschreibung der Codeerzeugung, die für die Optimierung von Programmen und das Verständnis des Compilers unumgänglich ist. Weiterhin enthält das Buch die Übersetzung der Referenz der wichtigsten Units der Laufzeitbibliothek von Free Pascal. Alle Teile des Buchs wurden nach Fertigstellung der Übersetzung noch einmal vom Begründer des Free-Pascal-Projekts, Florian Klämpfl, geprüft.

Autor:

Michaël Van Canneyt ist langjähriger Free-Pascal-Core-Entwickler und zeichnet, außer daß er der ursprüngliche Maintainer der Linux-Version ist und sich heute vorwiegend um die Klassenbibliotheken kümmert, auch für die Dokumentation des Projekts verantwortlich. Die vorliegende Dokumentation, die im englischen Original mehrere Tausend Seiten umfaßt, entstand im Laufe mehrerer Jahre. Sie wird ständig an die neuen Funktionen des Compilers angepaßt.

Der Autor ist Belgier, professioneller Software-Entwickler und lebt mit seiner Familie in Leuven. In seiner knapp bemessenen Freizeit schreibt er außer Software auch Fachbeiträge für die Zeitschrift Toolbox und widmet sich Buchprojekten. Er versteht außer Flämisch, Französisch, Englisch und Russisch auch ausgezeichnet Deutsch.

Free Pascal 2

Online-Daten

Michaël Van Canneyt

Übersetzung aus dem Englischen von
Jörg Braun, Michael Keßler, Florian Klämpfl, Christopher Özbek
und Rosa Riebl

Bibliographische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliographische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Alle Rechte vorbehalten. Ohne ausdrückliche, schriftliche Genehmigung des Herausgebers ist es nicht gestattet, das Buch oder Teile daraus in irgendeiner Form durch Fotokopie, Mikrofilm oder ein anderes Verfahren zu vervielfältigen oder zu verbreiten. Dasselbe gilt für das Recht der öffentlichen Wiedergabe.

Der Verlag macht darauf aufmerksam, daß die genannten Firmen- und Markenzeichen sowie Produktbezeichnungen in der Regel marken-, patent-, oder warenzeichenrechtlichem Schutz unterliegen.

Die Herausgeber übernehmen keine Gewähr für die Funktionsfähigkeit beschriebener Verfahren, Programme oder Schaltungen.

1. Auflage 2009

© 2009 by C&L Computer und Literaturverlag
Zavelsteiner Straße 20, 71034 Böblingen
E-Mail: info@cul.de
WWW: http://www.CuL.de

Coverdesign: Hawa & Nöh, Neu-Eichenberg
Satz: C&L-Verlag
Druck: PUT i RB DROGOWIEC
Printed in Poland

Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt

ISBN 978-3-936546-53-8

INHALTSVERZEICHNIS

Kapitel 4
Referenz der RTL (Online-Daten)
Seite 7

4.16 Unit DOS O8
4.16.1 Konstanten, Typen und Variablen O8
4.16.2 Prozeduren und Funktionen O11
4.17 Unit Strings O24
4.18 Unit Sockets O32
4.18.1 Konstanten, Typen und Variablen O32
4.18.2 Prozeduren und Funktionen O42
4.19 Unit ipc O58
4.19.1 Konstanten, Typen und Variablen O58
4.19.2 Prozeduren und Funktionen O62
4.20 Unit Video O75
4.20.1 Schreiben eines eigenen Bildschirmtreibers O76
4.20.2 Konstanten, Typen und Variablen O80
4.20.3 Prozeduren und Funktionen O85
4.21 Unit Mouse O93
4.21.1 Konstanten, Typdeklarationen und Variablen O93
4.21.2 Prozeduren und Funktionen O94
4.22 Unit Keyboard O99
4.22.1 Spezielle Hinweise zu Unix O99
4.22.2 Tastaturtreiber schreiben O100
4.22.3 Konstanten, Typen und Variable O105
4.22.4 Prozeduren und Funktionen O108

Stichwortverzeichnis O109

KAPITEL 4 (ERGÄNZUNG)

REFERENZ DER RTL (ONLINE-DATEN)

Die folgenden Abschnitte enthalten die Dokumentation der Units DOS, Strings, IPC, Sockets, Keyboard, Video und Mouse von Free Pascal 2.2.x. Dieses Dokument ist eine Ergänzung zu den Daten des gedruckten Buchs und wie dieses urheberrechtlich geschützt. Die Datei wird als Probe- und Referenzabschnitt vom C&L-Verlag exklusiv auf seinen Webseiten zur Verfügung gestellt und darf nicht an anderer Stelle ohne vorherige schriftliche Genehmigung des Verlags zur Verfügung gestellt werden.

Beschrieben werden zuerst jeweils die vordefinierten Konstanten und Variablen, anschließend die Funktionen und Prozeduren mit ihren jeweiligen Parametern und Beispielen. Optionale Parameter stehen immer in eckigen Klammern. Teilweise sind Beispielprogramme gezeigt, die in einem Editor eingegeben und mit Free Pascal kompiliert werden können.

4.16 Unit DOS

Die Unit DOS bietet den Zugriff auf eine Reihe betriebssystemspezifischer Aufrufe für Dateien, auf das Dateisystem, sowie Datum und Uhrzeit. Mit der Ausnahme von Palm-OS ist diese Unit auf allen von Free Pascal unterstützten Plattformen verfügbar.
Die Unit wurde ursprünglich von Florian Klämpfl für DOS geschrieben, von Mark May auf Linux portiert und von Michaël Van Canneyt erweitert. Der (inzwischen veraltete) Amiga-Port wurde von Nils Sjoholm geschrieben.
Bei anderen Betriebssystemen als DOS geht ein Teil der Funktionalität der Unit verloren, da die Routinen entweder nicht implementiert werden können oder nutzlos sind. Sind die Funktionen tatsächlich implementiert – und nicht nur als Prozedurrumpf aus Gründen der Abwärtskompatibilität –, verhalten sie sich auf den verschiedenen Betriebssystemen gleich.
Die Unit DOS dient der Kompatibilität zum veralteten 16-Bit Turbo-Pascal-Compiler und wird nicht mehr aktiv weiterentwickelt; das Interface ist eingefroren und die Funktionen und Prozeduren werden nur noch für das Portieren alter Turbo-Pascal-Routinen gepflegt. Für das Entwickeln neuer und moderner Programme wird dringend empfohlen, anstelle dieser Unit die neuere Unit *SysUtils* einzubinden.
Die Unit DOS bindet die Unit *BaseUnix* ein.

4.16.1 Konstanten, Typen und Variablen

Konstanten

DATEIMODUS-KONSTANTEN

Die folgenden Konstanten werden im Feld *Mode* des Record *TextRec* benötigt. Sie geben Auskunft über den Dateimodus der Textein- und Textausgabe (siehe Tabelle O4.1).

Konstante	Beschreibung	Wert
fmclosed	Datei ist geschlossen.	0D7B0h
fminput	Datei ist nur zum Lesen.	0D7B1h
fmoutput	Datei ist nur zum Schreiben.	0D7B2h
fminout	Datei darf gelesen und geschrieben werden.	0D7B3h

Tabelle O4.1: Die Dateimodus-Konstanten

DATEIATTRIBUTE

Die Dateiattribute-Konstanten werden in *FindFirst* und *FindNext* für die Definition benötigt, nach welchen Dateitypen zusätzlich zu normalen Dateien (solchen ohne Attribute) gesucht werden soll. Diese Flags gelten außerdem in den Routinen *SetFAttr* und *GetFAttr* für das Setzen und Auslesen der Attribute von Dateien. Zusammengestellt sind sie in Tabelle O4.2.

Konstante	Beschreibung	Wert
readonly	Schreibschutzattribut.	01h
hidden	Attribut für versteckte Datei.	02h
sysfile	Attribut für Systemdatei.	04h
volumeid	Datei ist das Volume Label (Datenträgerkennung).	08h
directory	Attribut für Verzeichnisse.	10h
archive	Archivattribut.	20h
anyfile	Attribut, das auf alle Dateien paßt.	3Fh

Tabelle O4.2: Mögliche Dateiattribute

Längenbeschreibungen

Die Unit DOS enthält einige Beschreibungen für Längenangaben:

Konstante	Wert	Beschreibung
filerecnamelength	255	Maximallänge des Dateinamensanteils im Record *FileRec*
FileNameLen	255	Maximale Länge eines Dateinamens
TextRecBufSize	256	Größe des voreingestellten Puffers in *TextRec*.
TextRecNameLength	256	Maximale Länge eines Dateinamens in *TextRec*.

CPU-Flags

Die in der Unit definierten CPU-Flags werden nicht verwendet und sind nur auf Kompatibilitätsgründen zu alten 16-Bit-Programmen definiert:

Konstante	Wert	Bedeutung (Flag)
fauxiliary	0010h	Auxiliary Flag
fcarry	0001h	Carry Flag
foverflow	0800h	Overflow Flag
fparity	0004h	Parity Flag
fsign	0080h	Sign
fzero	0040h	Zero Flag

Typdeklarationen

Die folgenden Stringtypen sind für die Vereinfachung des Umgangs mit Dateinamen deklariert. Bei den modernen Betriebssystemen mit langen Dateinamen sind in der Datei *dos.h.inc* die Längenangaben auf die Konstante *FileNameLen* gesetzt, die auf 255 im Header der Datei festgelegt ist. Damit wird die alte Deklaration von Free Pascal 1.x und Turbo Pascal überschrieben:

```
ComStr  = String[127];// Für Befehlszeilen
PathStr = String[79]; // Für komplette Pfade von Dateinamen
DirStr  = String[67]; // Für Verzeichnisse und (DOS-)Laufwerksbezeichnungen
NameStr = String[8];  // Für Dateinamen
ExtStr  = String[4];  // Für Dateierweiterungen
```

Werden Dateien auf der Festplatte gesucht, wird der folgende Record mit Daten gefüllt:

```
SearchRec = packed record
  SearchPos  : TOff;
  SearchNum  : LongInt;
  DirPtr     : Pointer;
  SearchType : Byte;
  SearchAttr : Byte;
  Mode       : Word;
  Fill       : array[1..1] of Byte;
  Attr       : Byte;
  Time       : LongInt;
  Size       : LongInt;
  Reserved   : Word;
  Name       : String;
  SearchSpec : String;
  NamePos    : Word;
end;
```

FileRec ist für die interne Repräsentation von typisierten und untypisierten Dateien. Textdateien werden in den folgenden Typen gespeichert:

```
const
  filerecnamelength = 255;
type
```

```
  FileRec = packed record
    Handle,
    Mode    : LongInt;
    RecSize : SizeInt;
    _private: array[1..3*SizeOf(SizeInt)+5*SizeOf(Pointer)] of Byte;
    UserData: array[1..32] of Byte;
    name    : array[0..filerecnamelength] of Char;
  end;
```

Die Definition des Textpuffers:

```
const
  TextRecNameLength = 256;
  TextRecBufSize    = 256;

type
  TextBuf = array[0..TextRecBufSize - 1] of Char;
  TextRec = packed record
    Handle   : THandle;
    Mode     : LongInt;
    bufsize  : SizeInt;
    _private : SizeInt;
    bufpos   : SizeInt;
    bufend   : SizeInt;
    bufptr   : ^Textbuf;
    openfunc : Pointer;
    inoutfunc: Pointer;
    flushfunc: Pointer;
    closefunc: Pointer;
    UserData : array[1..32] of Byte;
    name     : array[0..TextRecNameLength - 1] of Char;
    buffer   : TextBuf;
  end;
```

Hinweis: Diese Deklaration ist nicht binärkompatibel zu ihrem Gegenstück aus Turbo Pascal, da sich die Größe der einzelnen Felder geändert hat.
Dies ist der Record der CPU-Register für den Aufruf der Prozedur *MsDos*:

```
{$IFDEF CPUI386}
{$IFNDEF HAS_REGISTERS}
Registers = packed record
  case i: Integer of
   0 : (ax, f1, bx, f2, cx, f3, dx, f4, bp, f5, si, f51, di, f6, ds, f7,
        es, f8, flags,fs,gs: Word);
   1 : (al, ah, f9, f10, bl, bh, f11, f12, cl, ch, f13, f14, dl, dh: Byte);
   2 : (eax, ebx, ecx, edx, ebp, esi, edi: LongInt);
end;
{$ENDIF HAS_REGISTERS}
{$ENDIF CPUI386}
```

Der Typ *DateTime* wird in *PackTime* und *UnPackTime* benutzt, um das Dateidatum mit *GetFTime* und *SetFTime* setzen oder ermitteln zu können:

```
DateTime = record
  Year : Word;
  Month: Word;
  Day  : Word;
  Hour : Word;
  Min  : Word;
  Sec  : Word;
end;
```

Variablen

```
DosError: Integer;
```

In der Variable *DosError* wird von den Prozeduren in der Unit DOS ein Fehler gespeichert. *DosError* kann die folgenden Werte annehmen:

Wert	Bedeutung
2	Datei nicht gefunden.
3	Pfad nicht gefunden.
5	Zugriff verweigert.
6	Ungültiges Handle.
8	Nicht genügend Speicher.
10	Ungültige Umgebung.
11	Ungültiges Format.
18	Zu viele Dateien.

Tabelle 04.3: Fehlerkennungen in der Unit DOS

Andere Werte sind möglich, aber nicht dokumentiert.

4.16.2 Prozeduren und Funktionen

AddDisk

```
function AddDisk(const path: String): Byte;
```

AddDisk fügt einen Dateinamen *path* zur internen Laufwerksliste hinzu und ist nur unter Linux implementiert. Die Laufwerksliste bestimmt, welches Laufwerk in den Befehlen *DiskFree* und *DiskSize* benutzt wird. *DiskFree* und *DiskSize* benötigen eine Datei auf dem ausgewählten Laufwerk, es wird beim Aufruf der Systemfunktion *statfs* benötigt. Die Namen werden aufeinanderfolgend hinzugefügt. Die vier ersten Einträge sind vordefiniert:

- ».« für das aktuelle Laufwerk,
- »/fd0/.« für das erste Floppy-Laufwerk.
- »/fd1/.« für das zweite Floppy-Laufwerk.
- »/« für die erste Festplatte.

Der erste Aufruf von *AddDisk* fügt deshalb einen Namen für die zweite Festplatte der Liste hinzu, der zweite Aufruf für das dritte Laufwerk und so weiter, bis 23 Laufwerke hinzugefügt wurden (entsprechend Laufwerke »D:« bis »Z:«).
Siehe auch: *DiskFree*, *DiskSize*

DiskFree

```
function DiskFree(drive: Byte): Int64;
```

DiskFree liefert die Anzahl der freien Byte auf dem Laufwerk. Der Parameter *Drive* gibt an, welches Laufwerk untersucht werden soll. Dieser Parameter entspricht 1 für Laufwerk »a:«, 2 für Laufwerk »b:« und so weiter. Der Wert 0 als Parameter übergeben, gibt den freien Speicher des aktuellen Laufwerks zurück. Normalerweise ist der freie Speicher die Größe der Festplattenblöcke multipliziert mit der Anzahl der freien Blökke auf der Festplatte.

Nur für Linux:
DiskFree und *DiskSize* benötigen eine Datei auf dem ausgewählten Laufwerk, dies verlangt die Systemfunktion *statfs*. Vier Dateinamen werden von der Unit DOS vorinitialisiert:

- ».« für das aktuelle Laufwerk,
- »/fd0/.« für das erste Floppy-Laufwerk.
- »/fd1/.« für das zweite Floppy-Laufwerk.
- »/« für die erste Festplatte.

Es können jedoch bis zu 26 verschiedene Laufwerke angegeben werden. Ein weiteres nicht vordefiniertes Laufwerk fügt die Prozedur *AddDisk* hinzu.
Fehler: Es wird -1 zurückgegeben, wenn ein Fehler auftritt oder eine falsche Laufwerksnummer zurückgegeben wurde.

```
program Example6;   (* Beispiel für die Funktionen DiskSize und DiskFree *)
uses
  Dos;
begin
  WriteLn('Die Partition ist ', DiskSize(0), ' Byte groß.');
  WriteLn('Aktuell sind ', DiskFree(0), ' Byte frei.');
end.
```

Listing 04.1: dosex/ex6.pp

DiskSize

```
function DiskSize(drive: Byte): Int64;
```

DiskSize liefert die absolute Größe eines Laufwerks in Byte. Der Parameter *Drive* gibt das zu untersuchende Laufwerk an. Dieser Parameter ist 1 für Laufwerk a:, 2 für Laufwerk b: und so weiter. Der Wert 0 liefert den Speicher des aktuellen Laufwerks.
Nur für unixartige Betriebssysteme:
Diskfree und *Disksize* benötigen eine Datei auf dem ausgewählten Laufwerk, da dies von der Systemfunktion *statfs* verlangt wird. Vier Dateinamen werden von der Unit DOS vorinitialisiert:

- ».« für das aktuelle Laufwerk,
- »/fd0/.« für das erste Floppy-Laufwerk.
- »/fd1/.« für das zweite Floppy-Laufwerk.
- »/« für die erste Festplatte.

Es können jedoch bis zu 26 verschiedene Laufwerke angegeben werden. Um ein weiteres nicht vordefiniertes Laufwerk hinzuzufügen, wird *AddDisk* aufgerufen.
Fehler: -1, wenn ein Fehler auftritt oder eine falsche Laufwerksnummer zurückgegeben wird.
Ein Beispiel ist unter *DiskFree* zu finden.
Siehe auch: *DiskFree* und *AddDisk*.

DosExitCode

```
function DosExitCode: Word;
```

DosExitCode enthält im niederwertigen Byte den Exitcode eines Programms, das mit dem *Exec*-Befehl ausgeführt wurde.
Siehe auch: *Exec*.

```
program Example5;   (* Beispiel für die Funktionen Exec und DosExitCode *)
uses
  Dos;
begin
{$ifdef Unix}
  WriteLn('Starte /bin/ls -la');
  Exec('/bin/ls','-la');
{$else}
  WriteLn('Starte Dir');
  Exec(GetEnv('COMSPEC'), '/C dir');
{$endif}
  WriteLn('Programme endete mit ExitCode ', Lo(DosExitCode));
end.
```

Listing 04.2: dosex/ex5.pp

DosVersion

```
function DosVersion: Word;
```

DosVersion gibt die Versionsnummer des Betriebssystems oder die Kernelversion zurück. Das höherwertige Byte enthält die jeweilige Unter- und das niederwertige Byte die Hauptversionsnummer.
Hinweis: Auf Systemen, deren Version mehr als zwei Zahlen enthalten, werden nur die ersten beiden zurückgegeben. Unter Linux finden beispielsweise nur die ersten beiden Nummern Anwendung, so daß beispielsweise eine Linuxversion 2.6.24 nur 2.6 ausgibt. Bei Betriebssystemen, die keinen speziellen Systemaufruf besitzen, um die Kernelversion mitzuteilen, erhält man eine Versionsnummer von 0.

```
program Example1;           (* Beispiel für die Funktion DosVersion *)

uses
  Dos;
var
  OS: String[32];
  Version: Word;
begin
  {$ifdef LINUX}   OS := 'Linux';   {$endif}
  {$ifdef FreeBSD} OS := 'FreeBSD'; {$endif}
  {$ifdef NetBSD}  OS := 'NetBSD';  {$endif}
  {$ifdef Solaris} OS := 'Solaris'; {$endif}
  {$ifdef QNX}     OS := 'QNX';     {$endif}
  {$ifdef DOS}     OS := 'Dos';     {$endif}
  (* Die Liste in diesem Beispiel ist nicht vollständig! *)
  Version := DosVersion;
  WriteLn('Current ', OS, ' version is ', Lo(Version), '.', Hi(Version));
end.
```

Listing 04.3: dosex/ex1.pp

DTToUnixDate

```
function DTToUnixDate(DT: DateTime): LongInt;
```

DTToUnixDate wandelt den DOS-DateTime *DT* in einen Unix-Zeitstempel um. *DTToUnixDate* ist eine interne Funktion für Unix-Plattformen, sie sollte in der Anwendungsentwicklung keine Anwendung finden.
Siehe auch: *UnixDateToDT*, *PackTime*, *UnpackTime*, *GetTime* und *SetTime*.

EnvCount

```
function EnvCount: LongInt;
```

EnvCount ermittelt die Anzahl der Umgebungsvariablen.
Siehe auch: *EnvStr* und *GetEnv*.

EnvStr

```
function EnvStr(Index: LongInt): String;
```

EnvStr gibt das Name=Wert-Paar an der Position *Index* aus der Liste der Umgebungsvariablen zurück. Der Index des ersten Paares ist Null.
Fehler: Die Länge ist auf 255 Zeichen begrenzt..
Siehe auch: *EnvCount* und *GetEnv*.

```
program Example13; (* Beispiel für die Funktionen EnvCount und EnvStr *)
uses
  Dos;
var
```

```
  i : LongInt;
begin
  WriteLn('Current Environment is:');
  for i := 1 to EnvCount do WriteLn(EnvStr(i));
end.
```

Listing 04.4: dosex/ex13.pp

EXEC

```
procedure Exec(const path: PathStr; const comline: ComStr);
```

Exec führt das Program in *Path* mit den Optionen, die in *ComLine* gegeben sind, aus. Nachdem das Program beendet wurde, kehrt die Prozedur zurück. Der Exit-Code kann durch die Funktion *DosExitCode* bestimmt werden.
Fehler: Fehler werden in *DosError* ausgegeben.
Ein Beispiel ist bei der Erläuterung zu *DosExitCode* gezeigt.
Siehe auch: *DosExitCode*.

FEXPAND

```
function FExpand(const path: PathStr): PathStr;
```

Beschreibung: *FExpand* erweitert den Parameter *path* zu einer absoluten Pfadangabe, bestehend aus Laufwerksbuchstabe (unter DOS, Windows, OS/2), Pfadangabe und Dateiname. Auf Dateisystemen, die zwischen Groß- und Kleinschreibung unterscheiden (wie Linux und Unix), bleiben die Dateinamen unverändert, sonst werden sie in Großbuchstaben konvertiert.
Siehe auch: *FSplit*.

```
program Example11;       (* Beispiel für die Funktion FExpand *)
uses Dos;
begin
  WriteLn('Der vollständige Name dieses Programms ist ',
          FExpand(ParamStr(0)));
end.
```

Listing 04.5: dosex/ex11.pp

FINDCLOSE

```
procedure FindClose(var f: SearchRec);
```

FindClose gibt alle zum Suchrecord *f* gehörenden Ressourcen frei. Der Aufruf gibt die von den Aufrufen *FindFirst* und/oder *FindNext* belegten Ressourcen wieder frei, wozu die Unix-Implementation der Unit DOS eine Tabelle mit offenen Verzeichnissen bereitstellt. Wenn die Tabelle voll ist, wird eines der Verzeichnisse geschlossen und ein weiteres geöffnet. Dieses System funktioniert gut und ist im Grund vergleichbar mit der ursprünglichen DOS-Implementation, ist aber verhältnismäßig langsam, wenn eine große Zahl von Suchrecords geöffnet wird.
Deshalb wurde, um das FindFirst/FindNext-System zu beschleunigen, der *FindClose*-Befehl implementiert, der einen nicht mehr benötigten Suchvorgang abschließt und das zugehörige Verzeichnis schließt. Wenn man einen *SearchRec* also nicht mehr benötigt, teilt man das der Unit DOS mit dem Aufruf *FindClose* mit, womit der *SearchRec* geschlossen und der Tabellenslot freigegeben wird.
Hinweis: Es wird empfohlen, unter Linux nach Dateien mit dem Befehl *Glob* zu suchen.
Fehler: Fehler werden über *DosError* bekanntgegeben.
Siehe auch: *FindFirst* und *FindNext*.

FindFirst

```
procedure FindFirst(const path: PathStr; attr: Word; var f: SearchRec);
```

FindFirst leitet die Suche nach der durch *path* spezifizierten Datei mit den Attributen *Attr* ein. Die Prozedur speichert alle benötigten Daten, um die Suche weiterzuführen, im Record *f*. *Path* kann Wildcard-Zeichen (das Zeichen * entspricht einer beliebig langen Folge von beliebigen Zeichen, ? entspricht einem einzigen beliebigen Zeichen) enthalten. Wird eine den Angaben entsprechende Datei gefunden, wird diese in *F* gespeichert und 0 in *DosError* gespeichert. Eine mit *FindFirst* initialisierte Suche kann mit *FindNext* fortgesetzt und *FindClose* beendet werden.
Unter OS/2 können keine zwei unterschiedlichen *FindFirst*-Aufruf parallel gestartet werden. Es muß immer zuerst der eine Suchlauf beendet werden, bevor der zweite gestartet werden kann. Hält man sich nicht an diese Regel, wird ein Laufzeitfehler Nummer 6 (ungültiger Dateihandle) ausgelöst.
Fehler: Fehler werden in *DosError* gemeldet.
Siehe auch: *FindNext, FindClose*, *SysUtils.FindFirst*, *SysUtils.FindNext*, *SysUtils.FindClose*.

```
program Example7; (* Beispiel für die Funktionen FindFirst und FindNext *)
uses              (* ältere Variante mit der Unit Dos!                  *)
  Dos;
var
  Dir: SearchRec;

begin
  FindFirst('*.*', archive, Dir);
  WriteLn('FileName' + Space(32), 'FileSize': 9);
  while DosError = 0 do begin
    Writeln(Dir.Name + Space(40 - Length(Dir.Name)), Dir.Size:9);
    FindNext(Dir);
  end;
  FindClose(Dir);
end.
```

Listing 04.6: dosex/ex7.pp

FindNext

```
procedure FindNext(var f: SearchRec);
```

FindNext erhält als Argument einen *SearchRec*-Record, der durch einen *FindFirst*-Befehl initialisiert wurde, und versucht die nächste Datei zu finden, die mit den dem *FindFirst*-Befehl übergebenen Kriterien übereinstimmt. Falls *DosError* von Null verschieden ist, wurde keine andere die Kriterien erfüllende Datei gefunden.
Fehler: *DosError* meldet Fehler.
Ein Beispiel befindet sich unter *FindFirst*.
Siehe auch: *FindFirst*, *FindClose, SysUtils.FindFirst*, *SysUtils.FindNext*, *SysUtils.FindClose*.

FSearch

```
function FSearch(path: PathStr; dirlist: String): PathStr;
```

FSearch sucht die Datei *Path* in allen Verzeichnissen, die in *DirList* angeführt werden. Die vollständige Pfadangabe der gefundenen Datei wird zurückgeliefert. *DirList* muß eine Liste von Verzeichnissen sein, die durch Strichpunkte getrennt werden. Falls keine Datei gefunden wurde, wird eine leere Zeichenfolge zurückgeliefert.
Hinweis**:** Unter unixartigen Systemen kann als Trenner in *DirList* auch der dort für Pfadtrennungen übliche Doppelpunkt angegeben werden.
Siehe auch: *FExpand*.

```
program Example10;      (* Beispiel für die Funktion FSearch *)
uses
  Dos;
var
  s: PathStr;
begin
  s := FSearch(ParamStr(1), GetEnv('PATH'));
  if s = '' then WriteLn(ParamStr(1),' nicht im Suchpfad gefunden')
            else Writeln(ParamStr(1),' im Suchpfad befinden unter ', s);
end.
```

Listing 04.7: dosex/ex10.pp

FSplit

```
procedure FSplit(path: PathStr; var dir: DirStr; var name: NameStr;
                 var ext: ExtStr);
```

FSplit zerlegt einen vollständigen Dateinamen *path* in die drei Teile Pfad (*dir*), Dateiname (*name*) und Suffix (*ext*). Unter Linux gilt als Endung die Folge der Buchstaben, die hinter dem letzten Punkt (».«) stehen. Bei DOS gibt es eine Ausnahme, wenn *LFNSupport=False* ist, denn dann ist die Endung alles hinter dem *ersten* Punkt.
Siehe auch: *FSearch*.

```
program Example12;       (* Beispiel für die Funktion FSplit *)
uses
  Dos;
var
  dir : DirStr;
  name: NameStr;
  ext:  ExtStr;
begin
  FSplit(ParamStr(1), dir, name, ext);
  WriteLn('Gesplittet wird ', ParamStr(1), ' in');
  WriteLn('Pfad  : ', dir);
  WriteLn('Name  : ', name);
  WriteLn('Endung: ', ext);
end.
```

Listing 04.8: dosex/ex12.pp

GetCBreak

```
procedure GetCBreak(var breakvalue: Boolean);
```

GetCBreak ermittelt den Status der [Strg]-[Pause]-Prüfung unter DOS und Amiga OS. Wenn *BreakValue False* ist, wird nur während E/A-Operationen auf die Eingabe von [Strg]-[Pause] getestet. Falls *BreakValue True* ist, findet eine solche Überprüfung bei jedem Systemaufruf statt. Auf allen anderen Plattformen ergibt *BreakValue* immer den Wert *True*.
Siehe auch: *SetCBreak*.

GetDate

```
procedure GetDate(var year: Word; var month: Word; var mday: Word;
                  var wday: Word);
```

GetDate liest das Systemdatum aus. *Year* ist eine Zahl im Bereich von 1980 bis 2099, *Month* gibt die Zahl des Monats an, während *mday* den Tag im Monat beschreibt. Der Wochentag des aktuellen Datums wird in *wday* zurückgegeben, wobei der Sonntag als Tag 0, Montag als Tag 1 und so weiter zählen.
Siehe auch: *GetTime* und *SetDate*.

```
program Example2; (* Beispiel für die Funktion GetDate *)
uses
  Dos;
const
  DayStr   : array[0.. 6] of String[3] = ('Son', 'Mon', 'Die',
                                          'Mit', 'Don', 'Fre', 'Sam');
  MonthStr : array[1..12] of String[3] = ('Jan', 'Feb', 'Mrz', 'Apr',
                                          'Mai', 'Jun', 'Jul', 'Aug',
                                          'Sep', 'Okt', 'Nov', 'Dez');
var
  Year, Month, Day, WDay: Word;

begin
  GetDate(Year, Month, Day, WDay);
  WriteLn('Aktuelles Datum');
  WriteLn(DayStr[WDay], ', ', Day, '. ', MonthStr[Month], ' ', Year, '.');
end.
```

Listing 04.9: dosex/ex2.pp

GetEnv

```
function GetEnv(envvar: String): String;
```

GetEnv liefert den Wert der Umgebungsvariablen *EnvVar*. Unter Linux muß *EnvVar* unter Berücksichtigung der Klein- und Großschreibung angegeben werden. Wenn die Umgebungsvariable *EnvVar* nicht definiert ist, wird ein leerer String zurückgegeben.
Bei einigen Betriebssystemen wie beispielsweise Unix muß die Groß- und Kleinschreibung bei der Suche nach *EnvVar* beachtet werden.
Siehe auch: *EnvCount* und *EnvStr*.

```
program Example14;    (* Beispiel für die Funktion GetEnv *)
uses
  Dos;
begin
  WriteLn('Der aktuelle PATH ist ', GetEnv('PATH'));
end.
```

Listing 04.10: dosex/ex14.pp

GetFAttr

```
procedure GetFAttr(var f; var attr: Word);
```

GetFAttr ermittelt die Dateiattribute der Dateivariable *F*. Diese kann eine typisierte, untypisierte oder Textdatei sein, die zugewiesen sein muß, aber nicht geöffnet sein darf. Die Attribute können mit den folgenden Konstanten überprüft werden:

- ReadOnly = $01
- Hidden = $02
- SysFile = $04
- VolumeId = $08
- Directory = $10
- Archive = $20

Linux/Unix kennt die folgenden Attribute:

- *Directory*
- *ReadOnly* (falls der aktuelle Prozeß keinen Zugriff auf die Datei hat).
- *Hidden* (für Dateinamen, die mit einem Punkt beginnen).

Fehler: Fehler werden in *DosError* gemeldet.
Siehe auch: *SetFAttr*.

```
program Example8;    (* Beispiel für die Funktion GetFAttr *)
uses
  Dos;
var
  Attr: Word;
  f   : File;
begin
  Assign(f, ParamStr(1));
  GetFAttr(f, Attr);
  WriteLn('File ', ParamStr(1), ' besitzt die Attribute ', Attr);
  if (Attr and archive)   <> 0 then WriteLn('- Archive');
  if (Attr and directory) <> 0 then WriteLn('- Directory');
  if (Attr and readonly)  <> 0 then WriteLn('- Read-Only');
  if (Attr and sysfile)   <> 0 then WriteLn('- System');
  if (Attr and hidden)    <> 0 then WriteLn('- Hidden');
end.
```

Listing 04.11: dosex/ex8.pp

GetFTime

```
procedure GetFTime(var f; var time: LongInt);
```

GetFTime ermittelt die Uhrzeit der letzten Dateiänderung. Die Funktion *UnPackTime* entschlüsselt die Zeit aus dem LongInt-Format. *F* muß eine Dateivariable sein, die zugewiesen und geöffnet wurde.
Fehler: Fehler werden in *DosError* gemeldet.
Siehe auch: *SetFTime*, *PackTime* und *UnPackTime*.

```
program Example9;   (* Beispiel für die Funktion GetFTime *)
uses
  Dos;

function LO(w: Word): String;
var
  s :String;
begin
  Str(w, s);
  if w < 10 then LO := '0' + s
            else LO := s;
end;

var
  f    : File;
  Time : LongInt;
  DT   : DateTime;
begin
  if Paramcount > 0 then Assign(f, ParamStr(1))
                    else Assign(f, 'ex9.pp');
  Reset(f);
  GetFTime(f, Time);
  Close(f);
  UnPackTime(Time, DT);
  Write('Die Datei ', ParamStr(1), ' wurde letztmalig geändert am ');
  Writeln(LO(DT.Month), '.', LO(DT.Day), '.', DT.Year,
             ' um ', LO(DT.Hour), ':', LO(DT.Min));
end.
```

Listing 04.12: dosex/ex9.pp

GetIntVec

```
procedure GetIntVec(intno: Byte; var vector: Pointer);
```

GetIntVec liefert die Adresse des Interruptvektors *IntNo*.
Dieser Aufruf macht gar nichts und ist nur aus Gründen der Abwärtskompatibilität vorhanden, moderne Betriebssysteme erlauben diesen direkten Zugriff auf die Hardware nicht.
Siehe auch: *SetIntVec*.

GetLongName

```
function GetLongName(var p: String): Boolean;
```

Diese Funktion ist nur in den Versionen für Go32v2 und Win32 von Free Pascal verfügbar. *GetLongName* ändert den Dateinamen *p* in einen langen Dateinamen. Das Ergebnis des API-Aufrufs ist der lange Dateiname des kurzen Dateinamens *p*. Die Funktion ergibt *True*, wenn der Aufruf erfolgreich war. Die Funktion wird nur beim DOS-Extender unter Windows 95 und höher benötigt.
Fehler: Kann der API-Aufruf nicht erfolgreich ausgeführt werden, meldet die Funktion *False* zurück.
Siehe auch: *GetShortName*.

GetMsCount

```
function GetMsCount: Int64;
```

GetMSCount liefert eine Zahl von Millisekunden seit einem bestimmten Zeitpunkt. Dieser Zeitpunkt ist implementationsabhängig. Wird für Timingoperationen benötigt, die Subtraktion zweier aufeinanderfolgender Aufrufe ergibt die Zahl der Millisekunden zwischen den beiden Aufrufen.
Diese Funktion ist nicht sehr genau, es werden stattdessen systemspezifische Aufrufe für Timerabfragen empfohlen.
Siehe auch: *GetTime*.

GetShortName

```
function GetShortName(var p: String): Boolean;
```

Diese Funktion ist nur in den Versionen für GO32V2 und Win32 von Free Pascal enthalten. *GetShortName* ändert die Dateinamensangabe in *p* über einen API-Aufruf in dessen 8+3-Gegenstück. Ergebnis ist der kurze Dateiname zum langen Dateinamen in *p*.
Die Funktion gibt *True* zurück, wenn der API-Aufruf erfolgreich war.
Diese Funktion wird nur beim DOS-Extender unter Windows 95 und höher benötigt.
Fehler: Ist der API-Aufruf nicht erfolgreich, meldet die Funktion *False* zurück.
Siehe auch: *GetLongName*.

GetTime

```
procedure GetTime(var hour: Word; var minute: Word; var second: Word;
                  var sec100: Word);
```

GetTime liefert die Systemzeit. *Hour* wird im 24-Stunden-System angegeben. Die Minuten werden in *minute*, die Sekunden in *second* und die Hundertstelsekunden in *sec100* zurückgegeben.
Hinweis: Bestimmte Betriebssysteme wie beispielsweise Amiga OS setzen das Feld *sec100* immer auf Null.
Siehe auch: *GetDate* und *SetTime*.

```
program Example3;     (* Beispiel für die Funktion GetTime *)
uses
  Dos;
```

```
function LO(w: Word): String;
var
  s: String;
begin
  Str(w, s);
  if w < 10 then LO := '0' + s else LO :=s;
end;

var
  Hour, Min, Sec, HSec: Word;
begin
  GetTime(Hour,Min,Sec,HSec);
  WriteLn('Die aktuelle Uhrzeit:');
  WriteLn(LO(Hour), ':', LO(Min), ':', LO(Sec));
end.
```

Listing 04.13: dosex/ex3.pp

GetVerify

```
procedure GetVerify(var Verify: Boolean);
```

GetVerify liefert den Status des DOS-Flags *Verify*. Ist es *True*, überprüft DOS Daten, die auf die Festplatte geschrieben werden, indem sie nach jedem Schreibzugriff unmittelbar erneut gelesen und die Prüfsummen verglichen werden. Ist *Verify False*, werden die Daten nicht überprüft.
Hinweis: Auf Nicht-DOS-Systemen (ausgenommen OS/2-Anwendungen unter reinem DOS), ist *Verify* immer *True*.
Siehe auch: *SetVerify*.

Intr

```
procedure Intr(IntNo: Byte; var Regs: Registers);
```

Intr führt einen Softwareinterrupt mit der Nummer *IntNo* (zwischen 0 und 255) aus, indem die Prozessorregister auf *Regs* gesetzt werden. Der Inhalt der Register vor Rückkehr der Interruptprozedur wird in *Regs* gespeichert.
Hinweise**:** Unten Nicht-DOS-System macht dieser Aufruf nichts.
Siehe auch: *MsDos*.

MsDos

```
procedure MSDos(var Regs: Registers);
```

MSDos führt einen Betriebssystemaufruf aus. Die Prozedur entspricht einem Aufruf des *Intr*-Befehls mit der Interruptnummer für den Betriebssystemaufruf.
Hinweis: Bei Nicht-DOS-Betriebssystemen macht dieser Aufruf gar nichts, unter DOS wird der Interrupt 21h aufgerufen.
Siehe auch: *Intr*.

PackTime

```
procedure PackTime(var t: DateTime; var p: LongInt);
```

PackTime konvertiert das Datum und die Zeit, die in *T* gespeichert sind, in ein gepacktes Format, das an die Datei *SetFTime* als Parameter übergeben werden kann.
Siehe auch: *SetFTime*, *FindFirst*, *FindNext* und *UnPackTime.*

```
program Example4; (* Beispiel für die Funktionen PackTime und UnPackTime *)
uses
  Dos;
var
  DT   : DateTime;
```

```
  Time : LongInt;
begin
  with DT do begin
     Year  := 2008;
     Month := 11;
     Day   := 11;
     Hour  := 11;
     Min   := 11;
     Sec   := 11;
  end;
  PackTime(DT, Time);
  WriteLn('Gepackte Zeit: ',Time);
  UnPackTime(Time,DT);
  WriteLn('Wieder ausgepackt:');
  with DT do begin
    WriteLn('Jahr    ', Year);
    WriteLn('Monat   ', Month);
    WriteLn('Tag     ', Day);
    WriteLn('Stunde  ', Hour);
    WriteLn('Minute  ', Min);
    WriteLn('Sekunde ', Sec);
  end;
end.
```

Listing 04.14: dosex/ex4.pp

SetCBreak

```
procedure SetCBreak(BreakValue: Boolean);
```

SetCBreak setzt den Status der Überprüfung der Eingabe von [Strg]-[Pause]. Wenn *BreakValue False* ist, wird nur bei E/A-Operationen der Zustand von [Strg]-[Pause] überprüft, ansonsten bei jedem Systembefehl.
Dieser Aufruf ist nur unter DOS und Amiga OS verfügbar, bei anderen Betriebssystemen macht er nichts.
Siehe auch: *GetCBreak*.

SetDate

```
procedure SetDate(year: Word; month: Word; day: Word);
```

SetDate legt das systeminterne Datum fest, wobei *Year* eine Zahl zwischen 1980 und 2099 ist. Auf Linux werden für diesen Aufruf Administrator-Rechte benötigt. Auf anderen Unix-Systemen ist der Aufruf derzeit wirkungslos.
Siehe auch: *GetDate* und *SetTime*.

SetFAttr

```
procedure SetFAttr(var f; attr: Word);
```

SetFAttr setzt die Dateiattribute der Dateivariable *F*, die eine typisierte, untypisierte oder Textdatei sein kann. *F* muß zugewiesen, darf aber nicht geöffnet sein. Die Attribute können eine Summe der folgenden Konstanten sein:

- ReadOnly = 01h
- Hidden = 02h
- SysFile = 04h
- VolumeId = 08h
- Directory = 10h
- Archive = 20h
- AnyFile = 3fh

Bei unixartigen Dateisystemen wie Linux ist die Prozedur zwar implementiert, bewirkt aber nichts.
Fehler: Fehler werden in *DosError* gemeldet.
Siehe auch: *GetFAttr*.

SetFTime

```
procedure SetFTime(var f; Time: LongInt);
```

SetFTime stellt die Zeit der letzten Änderung der Datei *f* auf *Time* ein. Die Zeit muß im gepackten Zeitformat von DOS vorliegen, das durch *PackTime* erzeugt werden kann. *F* muß zugewiesen, aber darf nicht geöffnet sein. Bei unixartigen Dateisystemen wie Linux ist die Prozedur zwar implementiert, macht aber nichts.
Fehler: Fehler werden in *DosError* gemeldet.
Siehe auch: *GetFTime*, *PackTime* und *UnPackTime*.

SetIntVec

```
procedure SetIntVec(intno: Byte; vector: Pointer);
```

SetIntVec setzt den Interruptvektor *IntNo* auf die Interrupt-Prozedur, auf die *Vector* verweist.
Diese Prozedur ist zwar aus Gründen der Abwärtskompatibilität implementiert, sie ist aber wirkungslos.
Siehe auch: *GetIntVec*.

SetTime

```
procedure SetTime(hour: Word; minute: Word; second: Word; sec100: Word);
```

SetTime setzt die Zeit der internen Systemuhr. Der Parameter *hour* wird im 24-Stunden-System ausgelesen. *minute*, *second* und *sec100* enthalten die Minuten, Sekunden und Hundertstel einer Sekunde der zu setzenden Zeitangaben.
Unter Linux benötigt dieser Aufruf root-Rechte, bei anderen unixartigen Betriebssystemen macht der Aufruf nichts.
Siehe auch: *GetTime* und *SetDate*.

SetVerify

```
procedure SetVerify(Verify: Boolean);
```

SetVerify setzt oder entfernt das DOS-Flag *Verify*. Wenn *Verify True* ist, prüft DOS Daten, die auf die Festplatte geschrieben wurden, indem es sie wieder einliest und die Prüfsummen vergleicht. Falls *Verify False* ist, werden die geschriebenen Daten nicht überprüft. Hinweis: Auf Nicht-DOS-Systemen (ausgenommen OS/2-Anwendungen unter reinem DOS), ist *Verify* immer *True*.
Siehe auch: *SetVerify*.

SwapVectors

```
procedure SwapVectors;
```

SwapVectors vertauscht den Inhalt der internen Interruptvektorentabelle mit dem aktuellen Inhalt der Interruptvektoren. Der Befehl sollte vor und nach einem *Exec*-Aufruf ausgeführt werden, damit dem aufgerufenen Programm die standardmäßigen Interruptvektoren zur Verfügung stehen.
Hinweis: Bei diversen Betriebssystemen ist diese Prozedur nur als leerer Rumpf implementiert.
Siehe auch: *Exec* und *SetIntVec*.

UnixDateToDt

```
procedure UnixDateToDt(SecsPast: LongInt; var Dt: DateTime);
```

DTToUnixDate konvertiert den Unix-Zeitstempel in *SecsPast* in eine gültige *DateTime* in *DT*. Dies ist eine interne Funktion, die auf Unix-Plattformen implementiert ist und nicht verwendet werden sollte.
Siehe auch: *DTToUnixDate*, *PackTime*, *UnpackTime*, *GetTime* und *SetTime*.

UnpackTime

```
procedure UnpackTime(p: LongInt;var t: DateTime);
```

UnPackTime konvertiert die Zeit der letzten Änderung einer Datei *p* in einen *DateTime*-Datensatz. Die Zeit der letzten Dateiänderung finden die Funktionen *GetFTime*, *FindFirst* und *FindNext*.
Ein Beispiel ist bei *PackTime* gezeigt
Siehe auch: *GetFTime*, *FindFirst*, *FindNext* und *PackTime*.

Weekday

```
function Weekday(y: LongInt; m: LongInt; d: LongInt): LongInt;
```

WeekDay gibt die Nummer des Wochentags zurück, auf den das Datum d/m/y fällt. Sonntag ergibt dabei den Wert 0 und Samstag den Wert 6.
Fehler: Bei einem Fehler wird der Wert -1 zurückgegeben.
Siehe auch: *PackTime*, *UnpackTime*, *GetTime* und *SetTime*.

4.17 Unit Strings

Die Unit *Strings* von Free Pascal ist systemunabhängig und funktioniert auf allen unterstützten Plattformen. Alle Funktionen dieser Unit sind in erweiterten Varianten auch in der Unit *SysUtils* verfügbar, wobei die Syntax der Funktionen dann immer gleich ist. Das zeigt sich auch in den Beispielen, bei denen nur der Import anders lautet.
Die Routinen in dieser Unit kennen weder AnsiStrings noch WideStrings und auch keine nationalen Sonderzeichen. Alle Strings sind klassische Pascal-ShortStrings mit einer Länge von maximal 255 Zeichen. Für AnsiStrings und WideStrings muß auf die entsprechenden Routinen der Unit *SysUtils* gewechselt werden. In dieser Unit sind bei Längenangaben Definitionen des Datentyps *SizeInt* implementiert. Dieser Datentyp, der in der Unit *System* definiert ist, ist auf 32-Bit-Systemen ein Synonym für *LongInt*, auf 64-Bit-Systemen für *Int64*. Bei Strings sollte immer mit *SizeInt* gearbeitet werden.

4.17.1 Prozeduren und Funktionen

StrAlloc

```
function StrAlloc(L: SizeInt): PChar;
```

StrAlloc reserviert Speicher für den String mit der Länge *Len* auf dem Heap, das abschließende ASCII-#0 ist in dieser Längenangabe enthalten. Dann wird ein Zeiger auf den Speicherplatz zurückgegeben.
Fehler: Ist nicht genug Speicher für die Operation auf dem Heap vorhanden, wird ein Laufzeitfehler ausgelöst.
Hinweis: *Strings.StrAlloc* und *SysUtils.StrAlloc* (siehe dort) sind nicht kompatibel.
Siehe auch: *StrNew*, *StrPCopy* und *SysUtils.StrAlloc*.

StrCat

```
function StrCat(dest: PChar; source: PChar): PChar;
```

Hängt *Source* an *Dest* und gibt *Dest* zurück.
Fehler: Keine (es findet keine Längenüberprüfung statt).
Siehe auch: *StrLCat* und *SysUtils.StrCat*.

```
program Example11;          (* Beispiel für die Funktion StrCat *)
uses
  Strings;
const
  P1: PChar = 'Das ist ein PChar-String.';
var
  P2: PChar;
begin
  P2 := StrAlloc(StrLen(P1) * 2 + 1);
  StrMove(P2, P1, StrLen(P1) + 1);  (* P2 = P1                  *)
  StrCat(P2, P1);                   (* P2 noch einmal anhängen *)
  WriteLn('P2: ', P2);
  StrDispose(P2);
end.
```

Listing 04.15: stringex/ex11.pp

StrComp

```
function StrComp(Str1: PChar; Str2: PChar): SizeInt;
```

Vergleicht die beiden nullterminierten Strings *Str1* und *Str2*. Das Ergebnis ist

- ein negativer *SizeInt*, wenn *Str1* kleiner als *Str2* ist.
- 0, wenn *Str1* und *Str2* identisch sind.
- Ein positiver *SizeInt*, wenn *Str1* größer als *Str2* ist.

Ein Beispiel ist bei der Funktion *StrLComp* gezeigt.
Siehe auch: *StrLComp*, *StrIComp* und *StrLIComp*.

StrCopy

```
function StrCopy(Dest: PChar; Source: PChar): PChar;
```

Kopiert den nullterminierten String in *Source* nach *Dest* und gibt einen Zeiger auf *Dest* zurück. *Dest* muß genug freien Speicher reserviert haben, um *Source* aufnehmen zu können, mindestens aber *StrLen(Source)* + 1 Byte.
Fehler: Es wird keine Längenüberprüfung durchgeführt.
Siehe auch: *StrPCopy*, *StrLCopy*, *StrECopy* und *SysUtils.StrCopy*.

```
program Example4;        (* Beispiel für die Funktion StrCopy *)
uses
  Strings;
const
  P: PChar = 'Das ist ein PChar-String.';
var
  PP: PChar;
begin
  PP: = StrAlloc(StrLen(P) + 1);
  StrCopy(PP, P);
  if StrComp(PP, P) <> 0 then
    WriteLn('Oh-oh, Probleme ...')
  else
    WriteLn('Alles funktioniert: PP=', PP);
  StrDispose(PP);
end.
```

Listing 04.16: stringex/ex4.pp

StrDispose

```
procedure StrDispose(P: PChar);
```

Entfernt den String, auf den *P* zeigt, vom Heap und gibt den Speicher wieder frei.
Siehe auch: *StrNew* und *SysUtils.StrDispose*.

```
program Example17;       (* Beispiel für die Funktion StrDispose *)
uses
  Strings;

const
  P: PChar = 'Das ist ein PChar-String.';
var
  P2: PChar;

begin
  P2 := StrNew(P1);
  WriteLn('P2: ', P2);
  StrDispose(P2);
end.
```

Listing 04.17: stringex/ex17.pp

StrECopy

```
function StrECopy(Dest: PChar; Source: PChar): PChar;
```

Kopiert den nullterminierten String in *Source* nach *Dest* und gibt einen Zeiger auf das Stringende (also das terminierende ASCII #0) des kopierten Strings zurück.
Fehler: Es wird keine Längenüberprüfung durchgeführt.

Siehe auch: *StrLCopy*, *StrCopy* und *SysUtils.StrECopy*.

```
program Example6;     (* Beispiel für die Funktion StrECopy *)
uses
  Strings;
const
  P: PChar = 'Das ist ein PChar-String.';
var
  PP: PChar;
begin
  PP := StrAlloc(StrLen(P) + 1);
  if SizeInt(StrECopy(PP, P)) - SizeInt(PP) <> StrLen(P) then
    WriteLn('Hier ist was falsch!')
  else
    WriteLn('PP= ', PP);
  StrDispose(PP);
end.
```

Listing 04.18: stringex/ex6.pp

StrEnd

```
function StrEnd(p: PChar): PChar;
```

Liefert einen Zeiger auf das Ende von *p*, also auf das terminierende ASCII-#0-Zeichen.
Siehe auch: *StrLen* und *SysUtils.StrEnd*.

```
program Example6; (* Beispiel für die Funktion StrEnd *)
uses
  Strings;
const
  P: PChar = 'Das ist ein PChar-String';
begin
  if SizeInt(StrEnd(P)) - SizeInt(P) <> StrLen(P) then
    WriteLn('Da ist was falsch!')
  else
    WriteLn('Alles in Ordnung.');
end.
```

Listing 04.19: stringex/ex7.pp

StrIComp

```
function StrIComp(Str1: PChar; Str2: PChar): SizeInt;
```

Vergleicht die beiden nullterminierten Strings *Str1* und *Str2* und ignoriert dabei die Groß- und Kleinschreibung. Das Ergebnis ist

- ein negativer *SizeInt*, wenn *Str1* kleiner als *Str2* ist.
- 0, wenn die beiden Zeichenketten gleich sind.
- Ein positiver *SizeInt*, wenn *Str1* größer als *Str2* ist.

Siehe auch: *StrLComp*, *StrComp*, *StrLIComp* und *SysUtils.StrIComp*.

```
program Example8; (* Beispiel für die Funktion StrlComp *)
uses
  Strings;
const
  P1: PChar = 'Das ist der erste String.';
  P2: PChar = 'Das ist der zweite String.';
var
  L: SizeInt;
begin
  Write('P1 und P2 sind ');
  if StrComp(P1, P2) <> 0 then Write('NICHT ');
```

```
  Write('gleich. Der erste ');
  L := 1;
  while StrLComp(P1, P2, L) = 0 do Inc(L);
  Dec(l);
  WriteLn(l, ' Zeichen sind gleich');
end.
```

Listing 04.20: stringex/ex8.pp

StrLCat

```
function StrLCat(Dest: PChar; Source: PChar; l: SizeInt): PChar;
```

Fügt *L* Zeichen an von *Source* an *Dest* an und dann das abschließende ASCII-#0-Zeichen. Die Funktion gibt *Dest* zurück.
Siehe auch: *StrCat* und *SysUtils. StrLCat*.

```
program Example12;           (* Beispiel für die Funktion StrLCat *)
uses
  Strings;
const
  P1: PChar = '1234567890';
var
  P2: PChar;

begin
  P2:  = StrAlloc(StrLen(P1) * 2 + 1);
  P2^ := #0;                          // Länge Null
  StrCat(P2, P1);
  StrLCat(P2, P1, 5);
  WriteLn('P2 = ', P2);
  StrDispose(P2);
end.
```

Listing 04.21: stringex/ex12.pp

StrLComp

```
function StrLComp(Str1: PChar; Str2: PChar; L: SizeInt): SizeInt;
```

Vergleicht maximal *L* Zeichen der beiden nullterminierten Strings *Str1* und *Str2*. Das Ergebnis ist

- ein negativer *SizeInt*, wenn *Str1 < Str2*.
- 0, wenn *Str1 = Str2*.
- Ein positiver *SizeInt*, wenn *Str1 > Str2*.

Siehe auch: *StrComp*, *StrIComp*, *StrLIComp* und *SysUtils.StrLComp*.

```
program Example8;            (* Beispiel für die Funktion StrLComp *)
uses
  Strings;
const
  P1: PChar = 'Das ist der erste String.';
  P2: PChar = 'Das ist der zweite String.';
var
   L: SizeInt;

begin
  Write('P1 und P2 sind ');
  if StrComp(P1, P2) <> 0 then
    Write('NICHT');
  Write(' gleich. Die ersten ');
  L := 1;
```

```
  while StrLComp(P1, P2, L) = 0 do Inc(L);
  Dec(L);
  WriteLn(l, ' Zeichen sind identisch.');
end.
```

Listing 04.22: stringex/ex8.pp

StrLCopy

```
function StrLCopy(Dest: PChar; Source: PChar; MaxLen: SizeInt): PChar;
```

Kopiert *MaxLen* Zeichen von *Source* nach *Dest* und macht aus *Dest* einen nullterminierten String.
Fehler: Keine (es findet keine Längenüberprüfung statt).
Siehe auch: *StrCopy*, *StrECopy* und *SysUtils.StrLCopy*.

```
program Example5;          (* Beispiel für die Funktion StrLCopy *)
uses
  Strings;
const
  P: PChar = '123456789ABCDEF';
var
  PP: PChar;

begin
  PP := StrAlloc(11);
  WriteLn('Die ersten 10 Zeichen von P: ', StrLCopy(PP, P, 10));
  StrDispose(PP);
end.
```

Listing 04.23: stringex/ex5.pp

StrLen

```
function StrLen(p: PChar): SizeInt;
```

Liefert die Länge des nullterminierten Strings *p*.
Siehe auch: *StrNew* und *SysUtils.StrLen*.

```
program Example1;          (* Beispiel für die Funktion StrLen *)
uses
  Strings;
const
  P: PChar = 'Das ist ein PChar-Stringkonstante';

begin
  WriteLn('P: ', p);
  WriteLn('Länge(P): ', StrLen(P));
end.
```

Listing 04.24: stringex/ex1.pp

StrLIComp

```
function StrLIComp(Str1: PChar; Str2: PChar; L: SizeInt): SizeInt;
```

Vergleicht maximal *L* Zeichen der nullterminierten Strings *Str1* und *Str2* und ignoriert dabei die Groß- und Kleinschreibung. Das Ergebnis ist

- eine negative *SizeInt*-Zahl, wenn *Str1* < *Str2*.
- 0, wenn *Str1* und *Str2* gleich sind.
- Ein positiver *SizeInt*, wenn *Str1* > *Str2*.
- Ein Beispiel ist bei der Funktion *StrIComp* gezeigt.

Siehe auch: *StrLComp*, *StrComp*, *StrIComp* und *SysUtils.StrLIComp*.

StrLower

```
function StrLower(p: PChar): PChar;
```

Konvertiert *p* in Kleinbuchstaben. Das Funktionsergebnis ist *p*.
Siehe auch: *StrUpper* und *SysUtils.StrLower*.

```
program Example14; (* Beispiel für StrLower und StrUpper *)
uses
  Strings; { oder: SysUtils }

const
  P1: PChar = 'DAS IST EIN PCHAR-STRING IN GROSSBUCHSTABEN.';
  P2: PChar = 'das ist ein pchar-string in kleinbuchstaben';

begin
  WriteLn('Großschreibung:  ', StrUpper(P2));
  StrLower(P1);
  WriteLn('Kleinschreibung: ', P1);
end.
```

Listing 04.25: stringex/ex14.pp

StrMove

```
function StrMove(Dest: PChar; Source: PChar; L: SizeInt): PChar;
```

Kopiert *L* Zeichen von *Source* nach *Dest*. Dabei wird kein terminierendes Nullzeichen übertragen. Die Funktion gibt *Dest* zurück.
Siehe auch: *StrLCopy*, *StrCopy* und *SysUtils.StrMove*.

```
program Example10;      (* Beispiel für die Funktion StrMove *)
uses
  Strings;
const
  P1: PChar = 'Das ist ein PChar-String.';
var
  P2: PChar;

begin
  P2 := StrAlloc(StrLen(P1) + 1);
  StrMove(P2, P1, StrLen(P1) + 1); // P2 := P1
  WriteLn(' P2 = ', P2);
  StrDispose(P2);
end.
```

Listing 04.26: stringex/ex10.pp

StrNew

```
function StrNew(p: PChar): PChar;
```

Kopiert *p* auf den Heap und gibt einen Zeiger auf die Kopie zurück.
Fehler: Die Funktion gibt *NIL* zurück, wenn nicht genug Speicher für die Kopie verfügbar ist.
Siehe auch: *StrCopy*, *StrDispose* und *SysUtils.StrNew*.

```
program Example16;      (* Beispiel für die Funktion StrNew *)
uses
  Strings;
const
  P1: PChar = ' Das ist ein PChar-String';
var
  P2: PChar;
```

```
begin
  P2 := StrNew(P1);
  if P1 = P2 then WriteLn('Das kann nicht passieren ...')
             else WriteLn('P2: ', P2);
  StrDispose(P2);
end.
```

Listing 04.27: stringex/ex16.pp

StrPas

```
function StrPas(p: PChar): ShortString;
```

Konvertiert den nullterminierten String *p* in einen Pascal-String um und gibt ihn als Funktionsergebnis zurück. Der Ergebnisstring wird bei einer Länge von 255 Zeichen abgeschnitten.
Siehe auch: *StrPCopy* und *SysUtils.StrPas*.

```
program Example3;            (* Beispiel für die Funktion StrPas *)
uses
  Strings;

const
  P: PChar = 'Das ist ein PChar-String';
var
  S: String;

begin
  S := StrPas(P);
  WriteLn('S: ', S);
end.
```

Listing 04.28: stringex/ex3.pp

StrPCopy

```
function StrPCopy(d: PChar; const s: String): PChar;
```

Konvertiert den Pascal-String in *s* in einen nullterminierten String und kopiert ihn nach *d*. *d* muß groß genug sein, um den String aufnehmen zu können, das heißt, eine Größe von *Length(s) + 1* besitzen.
Fehler: Es wird keine Längenprüfung durchgeführt.
Siehe auch: *StrPas* und *SysUtils.StrPCopy*.

```
program Example2; (* Beispiel für die Funktion StrPCopy *)
uses
  Strings;

const
  S = 'das ist ein normaler String.';
var
  P: PChar;

begin
  P := StrAlloc(Length(S) + 1);
  if StrPCopy (P, S) <> P then
    WriteLn('Das ist unmöglich!!')
  else
    WriteLn(P);
  StrDispose(P);
end.
```

Listing 04.29: stringex/ex2.pp

StrPos

```
function StrPos(Str1: PChar; Str2: PChar): PChar;
```

Gibt einen Zeiger auf das erste Vorkommen von *Str2* in *Str1* zurück. Enthält *Str1* den gesuchten Teilstring nicht, wird *NIL* zurückgegeben.
Siehe auch: *StrScan*, *StrRScan* und *SysUtils.StrPos*.

```
program Example15; (* Beispiel für die Funktion StrPos *)
uses
  Strings; (* oder: SysUtils *)
const
  P: PChar = 'Das ist ein PChar-String.';
  S: PChar = 'ist';
begin
  WriteLn('Position von "ist" in P: ', SizeInt(StrPos(P, S)) - SizeInt(P));
end.
```

Listing 04.30: stringex/ex15.pp

StrRScan

```
function StrRScan(p: PChar; c: Char): PChar;
```

Gibt einen Zeiger auf das letzte Vorkommen des Zeichens *c* im nullterminierten String *p* zurück. Wird *c* nicht gefunden, liefert die Funktion den Wert *NIL*.
Ein Beispiel befindet sich bei der Funktion *StrScan*.
Siehe auch: *StrScan*, *StrPos* und *SysUtils.StrRScan*.

StrScan

```
function StrScan(p: PChar; c: Char): PChar;
```

Gibt einen Zeiger auf das erste Vorkommen des Zeichens *c* im nullterminierten String *p* zurück. Falls *c* nicht gefunden wird, liefert die Funktion den Wert *NIL*.
Siehe auch: *StrRScan*, *StrPos* und *SysUtils.StrScan*.

```
program Example13;  (* Beispiel für die Funktionen StrScan und StrRScan *)
uses
  Strings;
const
  P: PChar = 'Das ist ein PChar-String.';
  s: Char  = 's';
begin
  Writeln('P ab dem ersten  "s": ', StrScan(P,  s));
  Writeln('P ab dem letzten "s": ', StrRScan(P, s));
end.
```

Listing 04.31: stringex/ex13.pp

StrUpper

```
function StrUpper(p: PChar): PChar;
```

Konvertiert *p* in Großbuchstaben. Die Funktion gibt *p* zurück.
Ein Beispiel ist bei der Funktion *StrLower* gezeigt.
Siehe auch: *StrLower* und *SysUtils.StrUpper*.

4.18 Unit Sockets

Die Unit *Sockets* von Free Pascal wurde ursprünglich von Michaël Van Canneyt für Linux geschrieben und von Florian Klämpfl auf Windows portiert. Sie ruft folgende Units auf:

- baseunix
- UnixType

4.18.1 Konstanten, Typen und Variablen

Konstanten

Die folgenden Konstanten bestimmen die Socket-Domain, sie stehen im Aufruf von *Socket*.

Konstante	Wert	Adreßfamilie
AF_UNSPEC	0	Nicht angegeben.
AF_LOCAL	1	Unix-Sockets.
AF_UNIX	1	Unix-Domain-Sockets.
AF_INET	2	Internet-IP-Protokoll.
AF_AX25	3	Amateur-Radio AX.25.
AF_IPX	4	Novell IPX.
AF_APPLETALK	5	Appletalk DDP.
AF_NETROM	6	Amateur-Radio NetROM.
AF_BRIDGE	7	Multiprotocol Bridge.
AF_ATMPVC	8	ATM-PVCs.
AF_X25	9	Reserviert für das X.25-Projekt.
AF_INET6	10	IP Version 6.
AF_ROSE	11	Amateur-Radio X.25 PLP.
AF_DECnet	12	Reserviert für das DECnet-Project.
AF_NETBEUI	13	Reserviert für das 802.2LLC-Project.
AF_SECURITY	14	Sicherheits-Callback Pseudo AF.
AF_KEY	15	PF_KEY Key Management API.
AF_NETLINK	16	?
AF_ROUTE	AF_NETLINK	Alias zum Emulieren von 4.4BSD.
AF_PACKET	17	Packet-Familie.
AF_ASH	18	Ash.
AF_ECONET	19	Acorn Econet.
AF_ATMSVC	20	ATM SVCs.
AF_SNA	22	Linux SNA-Projekt.
AF_IRDA	23	IRDA-Sockets.
AF_PPPOX	24	PPPoX-Sockets.
AF_WANPIPE	25	Wanpipe-API-Sockets.
AF_LLC	26	Linux LLC.
AF_TIPC	30	TIPC-Sockets.
AF_BLUETOOTH	31	Bluetooth-Sockets.
AF_MAX	32	Maximalwert.

Die Unit definiert die folgenden Fehlerkonstanten (sind alles Alias-Angaben):

Fehlercode	Wert	Bedeutung
EsockEACCESS	ESysEAcces	Zugriff verboten.
EsockEBADF	EsysEBADF	Alias: Ungültiger Dateideskriptor.
EsockEFAULT	EsysEFAULT	Alias: Es ist ein Fehler aufgetreten.

EsockEINTR	EsysEINTR	Alias: Operation abgebrochen.
EsockEINVAL	EsysEINVAL	Alias: Ungültiger Wert angegeben.
EsockEMFILE	ESysEmfile	Fehlercode?
EsockEMSGSIZE	ESysEMsgSize	Fehlerhafte Meldungsgröße.
EsockENOBUFS	ESysENoBufs	Kein Pufferspeicher verfügbar.
EsockENOTCONN	ESysENotConn	Nicht verbunden.
EsockENOTSOCK	ESysENotSock	Dateideskriptor ist kein Socket.
EsockEPROTONOSUPPORT	ESysEProtoNoSupport	Protokoll wird nicht unterstützt.
EsockEWOULDBLOCK	ESysEWouldBlock	Operation würde blockieren.

Die folgenden Konstanten sind spezielle IP-Adressen, die häufig benötigt werden, wenn ein Socket an eine Schnittstelle auf der lokalen Maschine gebunden werden soll:

Konstante	Wert	Beschreibung
INADDR_ANY =	Cardinal(0)	Undokumentiert?
INADDR_NONE =	Cardinal($FFFFFFFF)	Undokumentiert?

Die folgenden Konstanten definieren die Protokolle:

Konstante	Wert	Beschreibung
IPPROTO_HOPOPTS	0	IPv6 Hop-by-Hop-Optionen.
IPPROTO_IP	0	Dummy-Protokoll für TCP.
IPPROTO_ICMP	1	Internet Control Message Protocol.
IPPROTO_IGMP	2	Internet Group Management Protocol.
IPPROTO_IPIP	4	IPIP-Tunnel (ältere KA9Q-Tunnel verwenden 94).
IPPROTO_TCP	6	Transmission Control Protocol.
IPPROTO_EGP	8	Exterior Gateway Protocol.
IPPROTO_PUP	12	PUP-Protokoll.
IPPROTO_UDP	17	User Datagram Protocol.
IPPROTO_IDP	22	XNS IDP Protokoll.
IPPROTO_TP	29	SO Transport Protocol Class 4.
IPPROTO_IPV6	41	IPv6-Header.
IPPROTO_ROUTING	43	IPv6-Routing-Header.
IPPROTO_FRAGMENT	44	IPv6 Fragmentation Header.
IPPROTO_RSVP	46	Reservation Protocol.
IPPROTO_GRE	47	General Routing Encapsulation.
IPPROTO_ESP	50	Encapsulating Security Payload.
IPPROTO_AH	51	Authentifizierungs-Header.
IPPROTO_ICMPV6	58	ICMPv6.
IPPROTO_NONE	59	IPv6, kein nächster Header.
IPPROTO_DSTOPTS	60	IPv6-Zieloptionen.
IPPROTO_MTP	92	Multicast Transport Protocol.
IPPROTO_ENCAP	98	Encapsulation Header.
IPPROTO_PIM	103	Protocol Independent Multicast.
IPPROTO_COMP	108	Compression Header Protocol.
IPPROTO_SCTP	132	Stream Control Transmission Protocol.
IPPROTO_MAX	255	Maximalwert für IPPROTO-Optionen.
IPPROTO_RAW	255	Rohe IP-Pakete.

Die folgenden Konstanten sind größtenteils undokumentierte Optionen für *GetSockOpt* und *SetSockOpt*. Alle vom Autor irgendwo als dokumentiert gefundenen Werte sind hier dokumentiert. Wer nach ihnen googelt, landet unweigerlich wieder in der Dokumentation von Free Pascal:

Konstante	Wert	Beschreibung
IPV6_ADDRFORM	1	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_ADD_MEMBERSHIP	IPV6_JOIN_GROUP	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_AUTHHDR	10	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_CHECKSUM	7	Undokumentierte Getsockopt-Option?
IPV6_DROP_MEMBERSHIP	IPV6_LEAVE_GROUP	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_DSTOPTS	4	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_HOPLIMIT	8	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_HOPOPTS	3	Undokumentierte Getsockopt-Option?
IPV6_IPSEC_POLICY	34	Undokumentierte Getsockopt-Option?
IPV6_JOIN_ANYCAST	27	Undokumentierte Getsockopt-Option?
IPV6_JOIN_GROUP	20	Undokumentierte Getsockopt-Option?
IPV6_LEAVE_ANYCAST	28	Undokumentierte Getsockopt-Option?
IPV6_LEAVE_GROUP	21	Undokumentierte Getsockopt-Option?
IPV6_MTU	24	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_MTU_DISCOVER	23	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_MULTICAST_HOPS	18	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_MULTICAST_IF	17	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_MULTICAST_LOOP	19	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_NEXTHOP	9	Undokumentierte Getsockopt-Option?
IPV6_PKTINFO	2	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_PKTOPTIONS	6	Undokumentierte Getsockopt-Option?
IPV6_PMTUDISC_DO	2	Immer DF.
IPV6_PMTUDISC_DONT	0	Nie DF-Frames senden
IPV6_PMTUDISC_WANT	1	Use per route hints.
IPV6_RECVERR	25	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_ROUTER_ALERT	22	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_RTHDR	5	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_RTHDR_LOOSE	0	Hop muß kein Nachbar sein.
IPV6_RTHDR_STRICT	1	Hop muß ein Nachbar sein.
IPV6_RTHDR_TYPE_0	0	IPv6-Routing-Header Typ 0.
IPV6_RXDSTOPTS	IPV6_DSTOPTS	Undokumentierte Getsockopt-Option?
IPV6_RXHOPOPTS	IPV6_HOPOPTS	Undokumentierte Getsockopt-Option?
IPV6_RXSRCRT	IPV6_RTHDR	Undokumentierte Getsockopt-Option?
IPV6_UNICAST_HOPS	16	Siehe *http://linux.die.net/man/7/ipv6*
IPV6_V6ONLY	26	Undokumentierte Getsockopt-Option?
IPV6_XFRM_POLICY	35	Undokumentierte Getsockopt-Option?

Für die folgende Liste von Konstanten gilt das selbe wie für die letzte Tabelle:

Konstante	Wert	Beschreibung
IP_ADD_MEMBERSHIP	35	IP-Gruppen-Mitgliedschaft hinzufügen.
IP_ADD_SOURCE_MEMBERSHIP	39	Der Quellgruppe beitreten.
IP_BLOCK_SOURCE	38	Daten von der Quelle blockieren.
IP_DEFAULT_MULTICAST_LOOP	1	Undokumentiert?
IP_DEFAULT_MULTICAST_TTL	1	Undokumentiert?
IP_DROP_MEMBERSHIP	36	IP-Gruppen-Mitgliedschaft verwerfen.
IP_DROP_SOURCE_MEMBERSHIP	40	Quellgruppe verlassen.
IP_HDRINCL	3	Header ist mit Daten enthalten.
IP_MAX_MEMBERSHIPS	20	Undokumentiert?
IP_MSFILTER	41	Undokumentiert?
IP_MTU_DISCOVER	10	Undokumentiert?
IP_MULTICAST_IF	32	set/get IP-Multicast-i/f.
IP_MULTICAST_LOOP	34	set/get IP-Multicast-Loopback.
IP_MULTICAST_TTL	33	set/get IP-Multicast-TTL.
IP_OPTIONS	4	IP-pro-Paket.
IP_PKTINFO	8	Undokumentiert?
IP_PKTOPTIONS	9	Undokumentiert?
IP_PMTUDISC	10	Undokumentiert?
IP_PMTUDISC_DO	2	Immer DF.
IP_PMTUDISC_DONT	0	Nie DF-Frames senden.
IP_PMTUDISC_WANT	1	Use per route hints.
IP_RECVERR	11	Undokumentiert?
IP_RECVOPTS	6	Alle IP-Optionen mit Datagram empfangen.
IP_RECVRETOPTS	IP_RETOPTS	IP-Optionen für Antwort empfangen.
IP_RECVTOS	13	Undokumentiert?
IP_RECVTTL	12	Undokumentiert?
IP_RETOPTS	7	Set/get-Optionen für IP pro Packet.
IP_ROUTER_ALERT	5	Undokumentiert?
IP_TOS	1	Type of Service und Precedence.
IP_TTL	2	TTL der IP (time to live).
IP_UNBLOCK_SOURCE	37	Daten von Quelle entblockieren.

Die folgenden Konstanten sind Multicast-Gruppenoptionen:

Konstante	Wert	Beschreibung
MCAST_EXCLUDE	0	Undokumentiert?
MCAST_INCLUDE	1	Undokumentiert?
MCAST_JOIN_GROUP	42	Quellgruppe beitreten.
MCAST_BLOCK_SOURCE	43	Block from given group
MCAST_UNBLOCK_SOURCE	44	Unblock from given group
MCAST_LEAVE_GROUP	45	Quellgruppe verlassen.
MCAST_JOIN_SOURCE_GROUP	46	Join source-spec group.
MCAST_LEAVE_SOURCE_GROUP	47	Leave source-spec group.
MCAST_MSFILTER	48	Undokumentiert?

Die folgenden Konstanten sind Optionen für das Senden und Empfangen von Messages mit Datagrammen:

Konstante	Wert	Beschreibung
MSG_CONFIRM	$0800	Send flags: Confirm connection.
MSG_CTRUNC	$0008	Receive flags: Control Data was discarded (buffer too small).
MSG_DONTROUTE	$0004	Send flags: don't use gateway.
MSG_DONTWAIT	$0040	Receive flags: Non-blocking operation request.
MSG_EOF	MSG_FIN	Alias for MSG_FIN.
MSG_EOR	$0080	Receive flags: End of record.
MSG_ERRQUERE	$2000	Receive flags: ?
MSG_FIN	$0200	Receive flags: ?
MSG_MORE	$8000	Receive flags: ?
MSG_NOSIGNAL	$4000	Receive flags: Suppress SIG_PIPE signal.
MSG_OOB	$0001	Receive flags: receive out-of-band data.
MSG_PEEK	$0002	Receive flags: peek at data, don't remove from buffer.
MSG_PROXY	$0010	Receive flags: ?
MSG_RST	$1000	Receive flags: ?
MSG_SYN	$0400	Receive flags: ?
MSG_TRUNC	$0020	Receive flags: packet Data was discarded (buffer too small).
MSG_TRYHARD	MSG_DONT-ROUTE	Receive flags: ?
MSG_WAITALL	$0100	Receive flags: Wait until operation completed.

Die folgenden typisierten Konstanten stellen besondere Internet-Adressen dar:

Konstante/Datentyp/Vorbelegung	Beschreibung
NoAddress: in_addr = (s_addr:0);	Konstante, die eine ungültige (keine) Netzwerkadresse mitteilt.
NoAddress6: in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0));	Konstante, die eine ungültige (keine) IPv6-Netzwerkadresse mitteilt.
NoNet: in_addr = (s_addr:0);	Konstante, die eine ungültige (keine) Netzwerkadresse mitteilt.
NoNet6: in6_addr = (u6_addr16: (0, 0, 0, 0, 0, 0, 0, 0));	Konstante, die eine ungültige (keine) IPv6-Netzwerkadresse mitteilt.

Die folgenden Konstanten legen die Protokollfamilie fest. Sie werden im Aufruf von der Prozedur *Socket* benötigt:

Konstante	Wert	Protokollfamilie
PF_APPLETALK	AF_APPLETALK	Appletalk DDP
PF_ASH	AF_ASH	Ash
PF_ATMPVC	AF_ATMPVC	ATM PVCs
PF_ATMSVC	AF_ATMSVC	ATM SVCs
PF_AX25	AF_AX25	Amateur Radio AX.25
PF_BLUETOOTH	AF_BLUETOOTH	Bluetooth Sockets
PF_BRIDGE	AF_BRIDGE	Multiprotocol Bridge
PF_DECnet	AF_DECnet	DECNET-Projekt
PF_ECONET	AF_ECONET	Acorn Econet
PF_INET	AF_INET	Internet IP-Protokoll
PF_INET6	AF_INET6	IP Version 6
PF_IPX	AF_IPX	Novell IPX
PF_IRDA	AF_IRDA	IRDA sockets
PF_KEY	AF_KEY	Key Management API
PF_LLC	AF_LLC	Linux LLC
PF_LOCAL	AF_LOCAL	Unix socket
PF_MAX	AF_MAX	Maximalwert
PF_NETBEUI	AF_NETBEUI	Reserviert für das 802.2LLC-Projekt
PF_NETLINK	AF_NETLINK	?
PF_NETROM	AF_NETROM	Amateur-Radio NetROM
PF_PACKET	AF_PACKET	Packet-Familie
PF_PPPOX	AF_PPPOX	PPPoX-Sockets
PF_ROSE	AF_ROSE	Amateur-Radio X.25 PLP
PF_ROUTE	AF_ROUTE	?
PF_SECURITY	AF_SECURITY	Sicherheitscallback Pseudo-PF
PF_SNA	AF_SNA	Linux SNA-Projekt
PF_TIPC	AF_TIPC	TIPC-Sockets
PF_UNIX	AF_UNIX	Unix Domain Sockets
PF_UNSPEC	AF_UNSPEC	Unspezifiziert
PF_WANPIPE	AF_WANPIPE	Wanpipe API-Sockets
PF_X25	AF_X25	Reserviert für das X.25-Projekt

Die folgenden Konstanten sind Optionen für das Datagram *SendMsg*:

Konstante	Wert	Beschreibung
SCM_SRCRT	IPV6_RXSRCRT	Undokumentierte Getsockopt-Option?
SCM_TIMESTAMP	SO_TIMESTAMP	Socket-Option: ?

Die folgenden Konstanten steuern das Herunterfahren von Sockets:

Konstante	Wert	Beschreibung
SHUT_RD	0	Leseanteil des Voll-Duplex-Sockets herunterfahren.
SHUT_WR	1	Schreibanteil des Voll-Duplex-Sockets herunterfahren.
SHUT_RDWR	2	Lese- und Schreibanteil des Voll-Duplex-Sockets herunterfahren.

Die folgenden Konstanten identifizieren verschiedene Socket-Typen, wie sie für die Routine *Socket* benötigt werden:

Konstante	Wert	Beschreibung
SOCK_STREAM	1	Socket-Typ: Stream (Verbindungs-) Sockettyp(TCP).
SOCK_DGRAM	2	Socket-Typ: Datagram (verbindungsloser) Socket (UDP).
SOCK_RAW	3	Socket-Typ: Roh-Socket.
SOCK_RDM	4	Socket-Typ: Nachricht wurde tatsächlich ausgeliefert.
SOCK_SEQPACKET	5	Socket-Typ: Sequentieller Packet-Socket.
SOCK_MAXADDRLEN	255	Maximale Socketadreßlänge für den Aufruf von Bind.

Die folgenden Konstanten sind Werte für die Socket-Optionen:

Konstante	Wert	Beschreibung
SOL_IP	0	Undokumentiert?
SOL_SOCKET	1	Socketoptionsebene: Socketebene.
SOL_IPV6	41	Socketebenen-Werte für IPv6: IPV6.
SOL_ICMPV6	58	Socketebenen-Werte für IPv6: ICMPV6.

Die folgende Konstante beschreibt ein System-Limit:

Konstante	Wert	Beschreibung
SOMAXCONN	128	Maximum queue length specifiable by listen.

Die folgenden Konstanten beschreiben Socket-Optionen:

Konstante	Wert	Beschreibung
S_IN	0	Eingabesocket im Socket-Paar.
S_OUT	1	Ausgabesocket im Socket-Paar.
SO_DEBUG	1	Socketoptionenebene: Debug.
SO_REUSEADDR	2	Socket-Option: Adresse wiederverwenden.
SO_TYPE	3	Socket-Option: Typ.
SO_ERROR	4	Socket-Option: Fehler.
SO_DONTROUTE	5	Socket-Option: Nicht routen.
SO_BROADCAST	6	Socket-Option: Broadcast.
SO_SNDBUF	7	Socket-Option: Puffer senden.
SO_RCVBUF	8	Socket-Option: Puffer empfangen.
SO_KEEPALIVE	9	Socket-Option: Keep-Alive.
SO_OOBINLINE	10	Socket-Option: ?
SO_NO_CHECK	11	Socket-Option: ?
SO_PRIORITY	12	Socket-Option: ?
SO_LINGER	13	Socket-Option: ?
SO_BSDCOMPAT	14	Socket-Option: ?
SO_PASSCRED	16	Socket-Option: ?
SO_PEERCRED	17	Socket-Option: ?
SO_RCVLOWAT	18	Socket-Option: ?
SO_SNDLOWAT	19	Socket-Option: ?
SO_RCVTIMEO	20	Socket-Option: ?
SO_SNDTIMEO	21	Socket-Option: ?

SO_SECURITY_AUTHENTICA-TION	22	Socket-Option: ?
SO_SECURITY_ENCRYPTION_-TRANSPORT	23	Socket-Option: ?
SO_SECURITY_ENCRYPTION_-NETWORK	24	Socket-Option: ?
SO_BINDTODEVICE	25	Socket-Option: ?
SO_ATTACH_FILTER	26	Socket-Option: ?
SO_DETACH_FILTER	27	Socket-Option: ?
SO_PEERNAME	28	Socket-Option: ?
SO_TIMESTAMP	29	Socket-Option: ?
SO_ACCEPTCONN	30	Socket-Option: ?

Die folgenden Konstanten sind Werte für TCP-Socketoptionen und nicht weiter beschrieben:

Konstante	Wert	Beschreibung
TCP_NODELAY	1	?
TCP_MAXSEG	2	?
TCP_CORK	3	?
TCP_KEEPIDLE	4	?
TCP_KEEPINTVL	5	?
TCP_KEEPCNT	6	?
TCP_SYNCNT	7	?
TCP_LINGER2	8	?
TCP_DEFER_ACCEPT	9	?
TCP_WINDOW_CLAMP	10	?
TCP_INFO	11	?
TCP_QUICKACK	12	?
TCP_CONGESTION	13	?
TCP_MD5SIG	14	?

Die folgenden Konstanten sind Werte für UDP-Socketoptionen und nicht weiter beschrieben:

Konstante	Wert	Beschreibung
UDP_CORK	1	?
UDP_ENCAP_ESPINUDP_NON_IKE	1	?
UDP_ENCAP_ESPINUDP	2	?
UDP_ENCAP_L2TPINUDP	3	?
UDP_ENCAP	100	?

Typdeklarationen

Die Unit definiert eine ganze Reihe spezialisierter Datentypen.

```
in6_addr  = packed record
  case Byte of
    0: (u6_addr8 : array[0..15] of Byte);
    1: (u6_addr16: array[0.. 7] of Word);
    2: (u6_addr32: array[0.. 3] of Cardinal);
```

```
    3: (s6_addr8 : array[0..15] of ShortInt);
    4: (s6_addr  : array[0..15] of ShortInt);
    5: (s6_addr16: array[0.. 7] of SmallInt);
    6: (s6_addr32: array[0.. 3] of LongInt);
end;
```

in6_addr ist ein Record für das Beschreiben einer allgemeinen IPv6-Adresse.

```
in_addr  = packed record
  case Boolean of
     true:  (s_addr : cuint32);         // inaddr_t=cuint32
     false: (s_bytes: packed array[1..4] of Byte);
end;
```

in_addr erfaßt die allgemeine Internet-Socket-Adresse.

```
linger = packed record
  l_onoff  : cint;
  l_linger : cint;
end;
```

Dieser Record wird im Aufruf von *setsockopt* benötigt, um *Linger*-Optionen zu definieren.

```
PIn6Addr = pin6_addr;
```

Zeiger auf den Datentyp *in6_addr*.

```
pin6_addr = ^in6_addr;
```

Zeiger auf den Datentyp *Tin6_addr*.

```
PInAddr = pin_addr
```

Ein Alias für *pin_addr*

```
PInetSockAddr = psockaddr_in
```

Zeiger auf den Datentyp *sockaddr_in*.

```
PInetSockAddr6 = psockaddr_in6
```

Zeiger auf den Datentyp *sockaddr_in6 type*.

```
pin_addr = ^in_addr
```

Zeiger auf den Record *in_addr*

```
plinger = ^linger
```

Zeiger auf den Datentyp *linger*.

```
psockaddr = ^sockaddr
```

Zeiger auf eine *TSockAddr*.

```
psockaddr_in = ^sockaddr_in
```

Zeiger auf den *sockaddr_in*.

```
psockaddr_in6 = ^sockaddr_in6
```

Zeiger auf *sockaddr_in6*.

```
psockaddr_un = ^sockaddr_un
```

Zeiger auf *sockaddr_un*.

```
sa_family_t = cushort
```

Datentyp für die Adreßfamilie.

```
sockaddr = packed record
  // if sa_len is defined, sa_family_t is smaller
  {$ifdef SOCK_HAS_SINLEN}
     sa_len: cuchar;
  {$endif}
  case Integer of
```

```
    0: (sa_family : sa_family_t;
        sa_data   : packed array[0..13] of cuint8);
    1: (sin_family: sa_family_t;
        sin_port  : cushort;
        sin_addr  : in_addr;
        sin_zero  : packed array[0..7] of cuint8);
end;
```

In *sockaddr* ist eine allgemeine Socketadresse für *Bind*, *Recv* und *Send* gespeichert.

```
sockaddr_in = packed record
  case boolean of
    false: (
       {$ifdef SOCK_HAS_SINLEN}sin_len: cuchar;{$endif}
       sin_family: sa_family_t;
       sin_port  : cushort;
       sin_addr  : in_addr;
       xpad      : array[0..7] of Char;// to get to the size of sockaddr...
    );
    true: (
     {$ifdef SOCK_HAS_SINLEN}len: cuchar;{$endif}
     family: sa_family_t;
     port  : cushort;
     addr  : cardinal;
     pad   : array[0..7] of Char; { to get to the size of sockaddr... }
    );
end;
```

sockaddr_in speichert eine INET-Socket-Adresse für die Aufrufe *Bind*, *Recv* und *Send*.

```
sockaddr_in6 = packed record
  Sin6_family   : sa_family_t;
  Sin6_port     : cuint16;
  Sin6_flowinfo : cuint32;
  Sin6_addr     : in6_addr;
  Sin6_scope_id : cuint32;
end;
```

Ein Alias für *sockaddr_in6*.

```
sockaddr_un = packed record
  sun_family: sa_family_t;
  sun_path  : array[0..107] of Char;
end;
```

sockaddr_un speichert eine Unix-Socket-Adresse für die Aufrufe *Bind*, *Recv* und *Send*.

```
TIn6Addr = in6_addr
```

Alias für den Datentyp *in6_addr*.

```
Tin6_addr = in6_addr
```

Alias für den Datentyp *sockaddr_in6*.

```
TInAddr = in_addr
```

Alias für den Record-Datentyp *in_addr*.

```
TInetSockAddr = sockaddr_in
```

Alias für den Datentyp *sockaddr_in*.

```
TInetSockAddr6 = sockaddr_in6
```

Alias für den Datentyp *sockaddr_in6*.

```
TIn_addr = in_addr
```

Alias für den Record-Datentyp *in_addr*.

```
TLinger = linger
```

Alias für den Datentyp *linger*.

```
TSockAddr = sockaddr
```

Alias für *sockaddr*.

```
TSockArray = array[1..2] of LongInt;
```

Datentyp, der vom Aufruf *SocketPair* zurückgegeben wird.

```
TSocket = LongInt
```

Alias für einfachere Kylix-Portierung.

```
TSockPairArray = array[0..1] of LongInt;
```

Ein Socket-Array für den Aufruf *SocketPair*.

```
TUnixSockAddr = packed record
  family: sa_family_t;
  path  : array[0..107] of Char;
end;
```

Alias für den Datentyp *sockaddr_un*.

4.18.2 Prozeduren und Funktionen

ACCEPT

```
function Accept(Sock: LongInt; var Addr;
                var Addrlen: LongInt): LongInt;
function Accept(Sock: LongInt; var addr: TInetSockAddr;
                var SockIn: File; var SockOut: File): Boolean;
function Accept(Sock: LongInt; var addr: TInetSockAddr;
                var SockIn: Text; var SockOut: Text) : Boolean;
function Accept(Sock: LongInt; var addr: String;
                var SockIn: Text; var SockOut: Text): Boolean;
function Accept(Sock: LongInt; var addr: String;
                var SockIn: File; var SockOut: File): Boolean;
```

Accept nimmt die Verbindung von einem Socket *Sock* an, der auf eine Verbindung wartet. Falls eine Verbindung akzeptiert wird, wird ein Dateideskriptor zurückgeliefert, bei einem Fehler -1. Über den zurückgegebenen Socket dürfen keine weiteren Verbindungen angenommen werden. Der übergebene Socket bleibt geöffnet.
Der *Accept*-Befehl speichert die Adresse des verbindenden Puffers (*entity*) in *Addr* und seine Länge in *Addrlen*. Vor dem Aufruf sollte *Addr* auf genügend Speicher verweisen und *Addrlen* sollte die Größe dieses Speichers vor dem Aufruf enthalten.
Die anderen Varianten des Aufrufs von *Accept* mit den *Text*- und *File*-Parametern sind Kombinationen aus der normalen Funktion *Accept* und anschließenden Aufrufen der Funktionen *Sock2Text* beziehungsweise *Sock2File*. Diese erweiterten Funktionen liefern bei erfolgreicher Ausführung ein *True* und sonst den Wert *False*.
Fehler: Bei einem Fehler wird von der einfachen Variante der Funktion der Wert -1 zurückgeliefert und *SocketError* auf einen der folgenden Werte gesetzt:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EOPNOTSUPP	Der Socket-Typ unterstützt keine Listen-Operation.
SYS_EFAULT	Addr zeigt auf einen Bereich außerhalb des Adreßraums.
SYS_EWOULDBLOCK	Die angefragte Operation würde den Prozeß blockieren.

Siehe auch: *Listen*, *Connect* und *Bind*.

```
program server;
(* program to test Sockets unit by Michaël van Canneyt and Peter Vreman
  Server Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
*)
uses Sockets;
var
  FromName : String;
  Buffer   : String[255];
  S        : LongInt;
  Sin, Sout: Text;
  SAddr    : TInetSockAddr;

procedure perror(const S: String);
begin
  WriteLn(S, SocketError);
  Halt(100);
end;

begin
  S := Socket(AF_INET, SOCK_STREAM, 0);
  if SocketError <> 0 then
    perror('Server: Socket: ');
  SAddr.Sin_family := AF_INET;
  (* port 50000 in network order: *)
  SAddr.Sin _port := htons(50000);
  SAddr.Sin_addr.s_addr := 0;
  if not Bind(S, SAddr, SizeOf(saddr)) then
    PError('Server: Bind: ');
  if not Listen(S, 1) then
    PError('Server: Listen: ');
  WriteLn('Waiting for Connect from Client, ' +
          'run now sock_cli in another tty');
  if not Accept(S, FromName, Sin, Sout) then
    PError('Server: Accept: ' + fromname);
  Reset(Sin);
  ReWrite(Sout);
  WriteLn(Sout, 'Message From Server');
  Flush(Sout);
  while not EoF(Sin) do begin
    ReadLn(Sin, Buffer);
    WriteLn('Server: Read: ', buffer);
  end;
end.
```

Listing 04.32: sockex/socksvr.pp

BIND

```
function Bind(Sock: LongInt; const Addr; AddrLen: LongInt): Boolean;
function Bind(Sock: LongInt; const addr: String): Boolean;
```

Bind verbindet den Socket *Sock* mit der Adresse *Addr*, die die Länge *Addrlen* hat. Die Funktion liefert *True*, falls sie erfolgreich war, ansonsten *False*. Die Variante des Bind-Befehls mit der *TUnixSockAddr* ist gleichwertig zum nacheinander folgenden Aufruf von *Str2UnixSockAddr* und der normalen *Bind*-Function. Die Funktion meldet *True*, wenn sie erfolgreich ausgeführt wurde, beim Auftreten eines Fehlers *False*.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_EINVAL	Der Socket ist bereits mit einer Adresse verbunden.
SYS_EACCESS	Die Adresse ist geschützt, das Progamm besitzt nicht die Berechtigung, sie zu öffnen.

Weitere mögliche Fehler können in den Unix-Manpages nachgelesen werden.
Siehe auch: *Socket*.

CLOSESOCKET

```
function CloseSocket(Sock: LongInt): LongInt;
```

CloseSocket schließt ein Socket-Handle. Die Funktion liefert 0 zurück, wenn der Socket erfolgreich geschlossen wurde.
Fehler: Bei einem Fehler wird der Wert -1 zurückgegeben.
Siehe auch: *Socket*.

CONNECT

```
function Connect(Sock: LongInt; const Addr; Addrlen: LongInt): Boolean;
function Connect(Sock: LongInt; const addr: TInetSockAddr;
                 var SockIn: Text; var SockOut: Text): Boolean;
function Connect(Sock: LongInt; const addr: TInetSockAddr;
                 var SockIn: File; var SockOut: File): Boolean;
function Connect(Sock: LongInt; const addr: String;
                 var SockIn: Text; var SockOut: Text): Boolean;
function Connect(Sock: LongInt; const addr: String;
                 var SockIn: File; var SockOut: File): Boolean;
```

Connect öffnet eine Verbindung zu einem Peer, dessen Adresse durch *Addr* beschrieben wird. *AddrLen* enthält die Länge der Adresse. Der Typ von *Addr* hängt von der Art der Verbinung ab, die zu etablieren versucht wird. Im allgemeinem ist der Typ jedoch *TSockAddr* oder *TUnixSockAddr*. Diese reguläre *Connect*-Funktion liefert einen Dateideskriptor, falls der Aufruf erfolgreich war.
Die anderen Varianten des Aufrufs von *Connect* mit den *Text*- und *File*-Parametern sind Kombinationen aus der normalen Funktion *Connect* und anschließenden Aufrufen der Funktionen *Sock2Text* beziehungsweise *Sock2File*. Diese erweiterten Funktionen liefern bei erfolgreicher Ausführung ein *True* und sonst den Wert *False*.
Fehler: Bei einem Fehler der normalen Funktion wird -1 zurückgeliefert und ein Fehlercode in *SocketError* gespeichert.
Siehe auch: *Listen*, *Bind* und *Accept*.

```
program Client;
(*
  Program to test Sockets unit by Michael van Canneyt and Peter Vreman
  Client Version, First Run sock_svr to let it create a socket and then
  sock_cli to connect to that socket
*)

uses
  Sockets;

procedure PError(const S: String);
begin
  WriteLn(S, SocketError);
  Halt(100);
end;
```

```
var
  SAddr    : TInetSpckAddr;
  Buffer   : String[255];
  S        : LongInt;
  Sin, Sout: Text;
  i        : Integer;
begin
  S := Socket(AF_UNIX, SOCK_STREAM, 0);
  if SocketError <> 0 then
   Perror('Client: Socket: ');
  SAddr.Sin_family := AF_INET;
  { port 50000 in network order }
  SAddr.Sin_port := htons(50000);
  { localhost: 127.0.0.1 in network order }
  SAddr.Sin_addr.s_addr := HostToNet((127 shl 24) or 1);
  if not Connect(S, SAddr, Sin, Sout) then
    PError('Client : Connect : ');
  Reset(Sin);
  ReWrite(Sout);
  Buffer := 'This is a textstring sent by the Client.';
  for i := 1 to 10 do WriteLn(Sout, Buffer);
  Flush(Sout);
  Readln(SIn,Buffer);
  WriteLn(Buffer);
  Close(Sout);
end.
```

Listing 04.33: sockex/sockcli.pp

```
program pfinger;
uses
  Sockets, Errors;
var
  Addr     : TInetSockAddr;
  S        : LongInt;
  Sin, Sout: Text;
  Line     : String;
begin
  Addr.family := AF_INET;
  { port 79 in network order }
  Addr.Sin_port := 79 shl 8;
  { localhost : 127.0.0.1 in network order }
  Addr.Sin_addr.s_addr := ((1 shl 24) or 127);
  S := Socket(AF_INET, SOCK_STREAM, 0);
  if not Connect(S, Addr, Sin, Sout) then begin
    WriteLn('Couldn''t connect to localhost');
    WriteLn('Socket error : ', strerror(SocketError));
    Halt(1);
  end;
  ReWrite(Sout);
  Reset(Sin);
  WriteLn(Sout, ParamStr(1));
  Flush(Sout);
  while not EoF(Sin) do begin
    ReadLn(Sin, line);
    WriteLn(line);
  end;
  Close(Sin);
```

```
  Close(Sout);
end.
```

Listing 04.34: Listing: sockex/pfinger.pp

FPACCEPT

```
function fpAccept(s: cint; Addrx: psockaddr; Addrlen: PSocklen): cint;
```

fpAccept nimmt eine Verbindung vom Socket *s* entgegen, der auf eine Verbindung gelauscht hat. Wird eine Verbindung akzeptiert, wird ein Dateideskriptor (eine positive Zahl) zurückgegeben, beim Auftreten eines Fehlers der Wert -1. Über den zurückgegebenen Socket dürfen keine weiteren Verbindungen angenommen werden. Für diesen Zweck bleibt der Original-Socket geöffnet.
Der Aufruf *fpAccept* füllt die Adresse in *Addrx* und setzt die Länge in *Addrlen*. *Addrx* sollte auf einen Bereich mit genügend Platz zeigen, *Addrlen* sollte auf die Größe des an dieser Stelle vor dem Aufruf befindlichen freien Speichers gesetzt werden.
Fehler: Bei einem Fehler werden -1 und der Fehler in *SocketError* zurückgegeben. Die folgenden Fehlerwerte können auftreten:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EOPNOTSUPP	Der Sockettyp unterstützt die Operation *Listen* nicht.
SYS_EFAULT	*Addr* zeigt aus dem gültigen Adreßbereich heraus.
SYS_EWOULDBLOCK	Die angeforderte Operation würde den Prozeß blockieren.

Siehe auch: *fpListen*, *fpConnect* und *fpBind*.
Für ein Listingbeispiel siehe *sockex/socksvr.pp* bei der Funktion *Accept*.

FPBIND

```
function fpBind(s: cint; Addrx: PSockaddr; Addrlen: TSocklen): cint;
```

fpBind bindet den Socket *s* an die Adresse *Addrx*. *Addrx* besitzt die Größe *Addrlen*. Die Funktion meldet 0, wenn sie erfolgreich ausgeführt wurde, und -1, wenn nicht.
Fehler: Fehler werden in *SocketError* zurückgegeben und enthalten die folgenden Werte:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_EINVAL	Der Socket ist bereits an eine Adresse gebunden.
SYS_EACCESS	Die Adresse ist geschützt und das Programm hat nicht das Recht, sie zu öffnen.

Weitere mögliche Fehler stehen in den Unix-Manpages.
Siehe auch: *Socket*.

FPCONNECT

```
function fpConnect(s: cint; name: PSockaddr; Namelen: TSocklen): cint;
```

fpConnect öffnet eine Verbindung zu einer Gegenstelle, deren Adresse in *Name* angegeben ist. *NameLen* enthält die Länge der Adresse. Der Typ von *Name* hängt von der Art der Verbindung, die zu öffnen versucht wird, ab und ist grundsätzlich entweder vom Typ *TSockAddr* oder *TUnixSockAddr*. Die Funktion *Connect* gibt bei Erfolg einen Dateideskriptor zurück.
Fehler: Bei einem Fehler ergibt die Funktion den Wert *-1*, Fehler werden in *SocketError* übergeben.
Siehe auch: *fpListen*, *fpBind* und *fpAccept*.

Listingbeispiele hierzu sind *sockex/sockcli.pp* und *sockex/pfinger.pp* bei der Funktion *connect*.

FPGETPEERNAME

```
function fpGetPeerName(s: cint; name: PSockaddr; Namelen: psocklen): cint;
```

fpGetPeerName gibt den Namen der an den angegebenen Socket *S* angebundenen Einheit zurück. Der Socket muß, damit dieser Aufruf funktioniert, verbunden sein.
Name sollte auf einen Speicherplatz zeigen, der groß genug ist, den Namen aufzunehmen, der verfügbare Platz wird in *Namelen* übergeben. Wurde die Funktion erfolgreich ausgeführt, enthält *Name* den Namen und *Namelen* dessen Länge.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOBUFS	Das System besitzt nicht genug Puffer, um die Operation durchführen zu können.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	*Addr* zeigt aus dem gültigen Adreßbereich heraus.
SYS_ENOTCONN	Der Socket ist nicht verbunden.

Siehe auch: *fpConnect*, *fpSocket*.

FPGETSOCKNAME

```
function fpGetSockName(s: cint; name: psockaddr; namelen: psocklen): cint;
```

fpGetSockName gibt den aktuellen Namen des angegebenen Sockets *s* zurück. *Name* sollte auf einen Speicherplatz mit ausreichend viel Platz zeigen, um den Namen aufnehmen zu können, die Größe des freien Speicherplatzes sollte in *Namelen* angegeben werden.
Kehrt die Funktion erfolgreich zurück, enthält *Name* den Namen und *Namelen* dessen Länge.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOBUFS	Das System besitzt nicht genug Puffer, um die Operation durchführen zu können.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	*Addr* zeigt auf einen Bereich außerhalb des eigenen Adreßbereichs.

Siehe auch: *fpBind*.

FPGETSOCKOPT

```
function fpGetSockOpt(s: cint; Level: cint; Optname: cint;
                      Optval: Pointer; Optlen: PSocklen): cint;
```

fpGetSockOpt erhält die Verbindungoption*Optname* für den Socket *S*. Der Socket kann auf unterschiedlichen Ebenen erlangt werden, die durch den Parameter *Level* festgelegt werden, der einen der folgendenWerte annehmen kann:

SOL_SOCKET	Vom Socket selbst.
XXX	Setzt Level auf XXX, die Protokollnummer des Protokolls, das die Option interpretieren soll.

Die Optionen werden an der in *Optval* angegebenen Speicherstelle abgelegt. *Optlen* sollte die ursprüngliche Länge von *Optval* enthalten und enthält bei der Rückkehr der Funktion die tatsächliche Größe der abgelegten Daten.
Bei Erfolg gibt die Funktion 0 zurück, bei einem Fehler den Wert -1.
Fehler: Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	*OptVal* zeigt auf einen Bereich außerhalb des eigenen Adreßbereichs.

Siehe auch: *fpSetSockOpt*.

FPLISTEN

```
function fpListen(s: cint; backlog: cint): cint;
```

fpListen lauscht auf bis zu *backlog* Verbindungen von Socket *s*. Der Socket *s* muß entweder vom Typ *SOCK_STREAM* oder vom Typ *Sock_SEQPACKET* sein.
Die Funktion gibt 0 zurück, wenn eine Verbindung angenommen wurde, -1 beim Auftreten eines Fehlers.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EOPNOTSUPP	Der Socket unterstützt die Operation *Listen* nicht.

Siehe auch: *fpSocket*, *fpBind* und *fpConnect*.

FPRECV

```
function fpRecv(s: cint; buf: Pointer; len: size_t; Flags: cint): ssize_t;
```

fpRecv liest maximal *len* Byte aus dem Socket *s* an die Adresse *buf*. Der Socket muß in einem verbundenen Status sein. *Flags* kann einer der folgenden Werte sein:

1	Daten außerhalb des Bands verarbeiten (Out-of band Data).
4	Das Routing soll umgangen und eine direkte Schnittstelle verwendet werden.
??	Warte auf die vollständige Anfrage oder generiere eine Fehlermeldung.

Die Funktion gibt die Zahl der aktuell aus dem Socket gelesenen Byte aus oder -1, wenn ein erkennbarer Fehler auftrat.
Fehler werden in *SocketError* gespeichert. Folgende Fehlercodes sind vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTCONN	Der Socket ist nicht verbunden.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Die Adresse ist außerhalb des eigenen Adreßbereichs.
SYS_EMSGSIZE	Die Meldung konnte nicht atomar gesendet werden.
SYS_EWOULDBLOCK	Die angeforderte Operation würde den Prozeß blockieren.
SYS_ENOBUFS	Das System besitzt nicht genug freie Puffer, um die Operation durchführen zu können.

Siehe auch: *Send*.

FPRECVFROM

```
function fpRecvFrom(s: cint;buf: Pointer; len: size_t; flags: cint;
                    from: PSockaddr; Fromlen: PSocklen): ssize_t;
```

fpRecvFrom empfängt aus dem Socket *s* Daten mit der maximalen Länge *Len* in den Puffer *Buf*. Der Empfang wird von den Optionen in *Flags* gesteuert. Der Speicherplatz, auf den *from* zeigt, wird mit der Adresse vom Sender gefüllt und die Länge in *Fromlen*. Die Funktion gibt die Zahl der empfangenen Byte aus.
Fehler: Tritt ein Fehler auf, wird -1 zurückgegeben.
Siehe auch: *fpSocket* und *fprecv*.

FPSEND

```
function fpSend(s: cint; msg: pointer; len: size_t; flags: cint): ssize_t;
```

fpSend sendet *Len* Byte ab der Adresse *Msg* an den Socket *s*. *s* muß dazu verbunden sein. Optionen können der Verbindung in *Flags* übergeben werden.
Die Funktion gibt die Zahl der gesendeten Byte zurück und -1, wenn ein erkennbarer Fehler auftrat. *flags* kann einer der folgenden Werte sein:

1	Daten außerhalb des Bandes verarbeiten (Out-of-band Data).
4	Das Routing soll umgangen und eine direkte Schnittstelle angesprochen werden.

Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Die Adresse ist außerhalb des eigenen Adreßbereichs.
SYS_EMSGSIZE	Die Meldung konnte nicht atomar gesendet werden.
SYS_EWOULDBLOCK	Die angeforderte Operation würde den Prozeß blockieren.
SYS_ENOBUFS	Das System besitzt nicht genug freien Puffer, um die Operation durchführen zu können.

Siehe auch: *fpRecv*.

FPSENDTO

```
function fpSendTo(s: cint; msg: pointer; len: size_t; flags: cint;
                  tox: psockaddr; tolen: tsocklen): ssize_t;
```

fpSendTo sendet Daten aus dem Puffer *Msg* mit der Länge *len* durch den Socket *s* mit der Option *Flags*. Die Daten werden an die Adresse *tox* gesandt, die die Länge *toLen* besitzt.
Fehler: Beim Auftreten eines Fehlers gibt die Funktion den Wert -1 zurück.
Siehe auch: *fpSocket*, *fpSend* und *fpRecvFrom*.

FPSETSOCKOPT

```
function fpSetSockOpt(s: cint; level: cint; optname: cint;
                      optval: pointer; optlen: tsocklen): cint;
```

fpSetSockOpt setzt die Verbindungsoptionen für den Socket *s*. Der Socket kann auf verschiedenen Ebenen verändert werden, wie im Parameter *level* angegeben. *level* kann einen der folgenden Werte annehmen:

SOL_SOCKET	Um den Socket selbst zu manipulieren.
XXX	Setzt Level auf XXX, die Nummer des Protokolls, das die Option interpretieren soll.

Die aktuelle Option wird im Puffer, auf den *optval* zeigt, gespeichert. Seine Länge wird mit *optlen* festgelegt.
Weitere Informationen zu diesem Aufruf erhält man in der Unix-Magepage zu *setsockopt*.
Fehler werden in *SocketError* gespeichert.
Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Die Adresse ist außerhalb des eigenen Adreßbereichs.

Siehe auch: *fpGetSockOpt*.

FPSHUTDOWN

```
function fpShutDown(s: cint; how: cint): cint;
```

fpShutDown schließt das eine Ende der zweiseitigen Verbindung in *s*. Der Parameter *how* beschreibt, wie die Verbindung heruntergefahren wird, und kann einer der folgenden Werte sein:

0	Weiterer Empfang ist nicht erlaubt.
1	Weiteres Senden ist nicht erlaubt.
2	Senden und Empfangen ist verboten.

Bei Erfolg gibt die Funktion den Wert 0 zurück, bei einem Fehler -1.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EBADF	Der Socket-Deskriptor ist ungültig.
SYS_ENOTCONN	Der Socket ist nicht verbunden.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.

Siehe auch: *fpSocket* und *fpConnect*.

FPSOCKET

```
function fpSocket(domain: cint; xtype: cint; protocol: cint): cint;
```

fpSocket erzeugt einen neuen Socket vom Typ *xType* mit dem Protokoll *Protocol* in der Domain *domain*. Domain, Socket-Typ und Protokoll können mit vordefinierten Konstanten belegt werden (siehe dazu den Abschnitt über Konstanten in diesem Kapitel ab Seite O32). Wird die Funktion erfolgreich abgeschlossen, liefert sie einen Socket-Deskriptor, der an einen nachfolgenden Aufruf von *fpBind* übergeben werden kann, beim Auftreten eines Fehlers gibt die Funktion -1 zurück.
Ein Beispiel ist bei der Funktion *Accept* gezeigt.
Fehler werden in *SocketError* gespeichert. Es sind folgende Fehlercodes vorgesehen:

SYS_EPROTONOSUPPORT	Der Protokolltyp oder das angebene Protokoll wird in dieser Domain nicht unterstützt.
SYS_EMFILE	Die prozeßbezogene Deskriptortabelle ist voll.
SYS_ENFILE	Die Dateitabelle des Systems ist voll.
SYS_EACCESS	Die Adresse ist geschützt und das Programm hat nicht das Recht, sie zu öffnen.
SYS_ENOBUFS	Das System besitzt nicht genug freie Puffer, um die Operation durchführen zu können.

Siehe auch: *SocketPair*.

FPSOCKETPAIR

```
function fpSocketPair(d: cint; xtype: cint; protocol: cint;
                      sv: pcint): cint;
```

fpSocketPair erzeugt zwei Sockets in der Domain *D* vom Typ *xType* mit dem Protokoll *Protocol*. Das Paar wird in *sv* zurückgegeben, die beiden Sockets unterscheiden sich nicht voneinander. Die Funktion gibt bei einem Fehler den Wert -1 zurück, geht alles glatt, meldet sie den Wert 0.
Fehler: Fehler werden in *SocketError* gespeichert, es sind die selben Codes wie bei der Funktion *Socket*.
Siehe auch: *Str2UnixSockAddr*.

GETPEERNAME

```
function GetPeerName(Sock: LongInt; var Addr;
                     var Addrlen: LongInt): LongInt;
```

GetPeerName liefert den Namen der Einheit, die mit dem Socket *Sock* verbunden ist. Damit dieser Aufruf gelingt, muß der Socket verbunden sein. *Addr* sollte auf genügend Speicher verweisen, um den Namen speichern zu können. Die Größe des Speichers, auf den *Addr* zeigt, sollte von *Addrlen* festgelegt werden. War die Funktion erfolgreich, wird *Addr* mit dem Namen gefüllt und *Addrlen* wird zur Länge von *Addr* gesetzt.
Fehler werden in *SocketError* gespeichert und können die folgenden Werte annehmen:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOBUFS	Das System hat nicht genügend Pufferspeicher, um die Operation auszuführen.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Addr zeigt auf einen Bereich außerhalb des Adreßraums.
SYS_ENOTCONN	Der Socket ist nicht verbunden.

Siehe auch: *Connect*, *Socket* und *connect*.

GETSOCKETNAME

```
function GetSocketName(Sock: LongInt; var Addr;
                       var Addrlen: LongInt) : LongInt
```

Hinweis: Diese Funktion ist veraltet.
GetSockName liefert den aktuellen Namen des angegebenen Sockets *Sock* zurück. *Addr* sollte auf genügend Speicher verweisen, um den Namen aufnehmen zu können. Die Größe des Speichers, auf den verwiesen wird, sollte in *Addrlen* angegeben werden. Wenn die Funktion erfolgreich war, wird *Addr* auf den Namen und *Addrlen* auf die Länge von *Addr* gesetzt.
Fehler: Fehler werden in *SocketError* gespeichert und können die folgenden Werte annehmen:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOBUFS	Das System hat nicht genügend Pufferspeicher, um die Operation auszuführen.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	*Addr* zeigt auf einen Bereich außerhalb des Adreßraums.

Siehe auch: *Bind*.

GETSOCKETOPTIONS

```
function GetSocketOptions(Sock: LongInt; Level: LongInt;
                          OptName: LongInt;
                          var OptVal;
                          var optlen: LongInt): LongInt;
```

GetSocketOptions liefert die Verbindungsoptionen für den Socket *Sock*. Diese Optionen können von verschiedenen Levels gewonnen werden. Der dafür vorgesehene Parameter *level* kann die folgenden Werte annehmen:

SOL_SOCKET	Der Socket selbst soll die Optionen bereitstellen.
XXX	Setzen Sie *Level* auf die Nummer des Protokolls, das die Option interpretieren soll.

Um mehr Informationen über diesen Befehl zu erhalten, schlagen Sie in den Unix-Manpages unter *getsockopt* nach.
Fehler: Fehler werden in *SocketError* gespeichert.
Folgende Werte sind möglich:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Addr zeigt auf einen Bereich außerhalb des Adreßraums.

Siehe auch: *SetSocketOptions*.

HOSTADDRTOSTR

```
function HostAddrToStr(Entry: in_addr): AnsiString;
```

HostAddrToStr wandelt die Hostadresse in *Entry* in einen menschenlesbaren String mit Punkttrennern um. Dabei handelt es sich prinzipiell um die selbe Funktion wie *NetAddrToStr*, aber mit den Byte in korrekter Reihenfolge.
Siehe auch: *NetAddrToStr*, *StrToHostAddr* und *StrToNetAddr*.

HOSTADDRTOSTR6

```
function HostAddrToStr6(Entry: Tin6_addr): AnsiString;
```

HostAddrToStr6 wandelt die IPV6-Hostadresse in *Entry* in einen menschenlesbaren String mit Punkttrennern um. Dabei handelt es sich prinzipiell um die selbe Funktion wie *NetAddrToStr6*, aber mit den Byte in korrekter Reihenfolge.
Siehe auch: *NetAddrToStr*, *StrToHostAddr*, *StrToNetAddr* und *StrToHostAddr6*.

HOSTTONET

```
function HostToNet(Host: in_addr): in_addr;
function HostToNet(Host: LongInt): LongInt;
```

HostToNet wandelt eine Hostadresse in eine Netzwerkadresse, wobei die Endianness der Hostmaschine berücksichtigt wird. Die Adresse kann mit Anführungszeichen und Punktrennern oder als LongInt angegeben werden.
Siehe auch: *NetToHost*, *NToHS*, *HToNS*, *ShortHostToNet* und *ShortNetToHost*.

HTONL

```
function htonl(host: LongInt): LongInt;
```

htonl sorgt dafür, daß die Byte in *host* für den Versand über das Netzwerk in der richtigen Reihenfolge stehen und gibt das richtig sortierte Ergebnis zurück.
Siehe auch: *htons*, *ntohl* und *ntohs*.

HTONS

```
function htons(host: Word): Word;
```

htons sorgt dafür, daß die Byte in *host* für den Versand über das Netzwerk in der richtigen Reihenfolge stehen und gibt das richtig sortierte Ergebnis zurück.
Siehe auch: *htonl*, *ntohl* und *ntohs*.

LISTEN

```
function Listen(Sock: LongInt; MaxConnect: LongInt): Boolean;
```

Listen wartet auf maximal *MaxConnect* Verbindungen vom Socket *Sock*, der vom Typ *SOCK_STREAM* oder *Sock_SEQPACKET* sein muß. Die Funktion liefert *true*, falls eine Verbindung angenommen wurde, *false* bei einem Fehler.
Fehler: Fehler werden in *SocketError* gespeichert. Folgende Werte sind möglich:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EOPNOTSUPP	Der Socket-Typ unterstützt keine Listen-Operation.

Siehe auch: *Socket*, *Bind* und *Connect*.

NETADDRTOSTR

```
function NetAddrToStr(Entry: in_addr): AnsiString;
```

NetAddrToStr in einen menschenlesbaren String mit Punkttrennern um.
Siehe auch: *HostAddrToStr*, *StrToNetAddr* und *StrToHostAddr*

NETADDRTOSTR6

```
function NetAddrToStr6(Entry: Tin6_addr): AnsiString;
```

NetAddrToStr6 wandelt die IPV6-Netzwerkadresse in *Entry* in einen lesbaren String um. Vom Grunde her ist das dasselbe wie *NetAddrToStr6*, allerdings sind hier die Daten in der richtigen Reihenfolge.
Siehe auch: *NetAddrToStr*, *StrToHostAddr*, *StrToNetAddr* und *StrToHostAddr6*.

NETTOHOST

```
function NetToHost(Net: in_addr): in_addr;
function NetToHost(Net: LongInt): LongInt;
```

NetToHost wandelt eine Netzwerkadresse in eine Hostadresse um, wobei die Endianness der Hostmaschine berücksichtigt wird. Die Adresse kann als punktgetrennter Block oder als LongInt-Zahl angegeben sein.
Siehe auch: *HostToNet*, *NToHS*, *HToNS*, *ShortHostToNet* und *ShortNetToHost*.

NTOHL

```
function NToHl(Net: LongInt): LongInt;
```

ntohs stellt sicher, daß die Bytes in *Net*, die aus dem Netzwerk erhalten wurden, in der richtigen Reihenfolge für die Verarbeitung auf der Hostmaschine sind, und gibt das Ergebnis in richtiger Reihenfolge aus.
Siehe auch: *htonl*, *htons* und *ntohs*.

NTOHS

```
function NToHs(Net: Word): Word;
```

ntohs stellt sicher, daß die Byte in *Net*, die aus dem Netzwerk erhalten wurden, in der richtigen Reihenfolge für die Verarbeitung auf der Hostmaschine sind, und gibt das Ergebnis in richtiger Reihenfolge aus.
Siehe auch: *htonl*, *htons* und *ntohl*.

RECV

```
function Recv(Sock: LongInt; var Buf; BufLen: LongInt;
              Flags: LongInt): LongInt;
```

Recv liest höchstens *Addrlen* Byte vom Socket *Sock* in die Adresse *Addr* ein. Es muß eine Verbindung zum Socket bestehen. *Flags* kann folgende Werte annehmen:

1	Daten außerhalb des Bands verarbeiten (Process out-of band data).
4	Routing umgehen und eine direkte Schnittstelle ansprechen.
??	Auf eine vollständige Anfrage oder die Meldung eines Fehlers warten.

Die Funktionen liefern die Anzahl an Byte, die tatsächlich vom Socket gelesen werden, oder -1, falls ein Fehler bemerkt wurde.
Fehler: Fehler werden in *SocketError* gespeichert. Es können folgende Werte auftreten:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTCONN	Der Socket ist nicht verbunden.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Die angegebene Adresse ist außerhalb der Adreßraums.
SYS_EMSGSIZE	Die Meldung kann nicht atomar (atomically) sein.
SYS_EWOULDBLOCK	Die angefragte Operation würde die Applikation blockieren.
SYS_ENOBUFS	Dem System steht nicht genug freier Pufferspeicher zur Verfügung.

Siehe auch: *Send*.

RECVFROM

```
function RecvFrom(Sock: LongInt; var Buf; Buflen: LongInt; Flags: LongInt;
                  var Addr; var AddrLen: LongInt): LongInt;
```

RecvFrom empfängt Daten aus dem Socket *Sock* in den Puffer *Buf* mit der Maximallänge *BufLen*. Der Empfang wird mit den Optionen in *Flags* kontrolliert. In *Addr* wird die Adresse des Senders eingetragen, die die Länge *AddrLen* haben darf. Die Funktion gibt die Zahl der empfangenen Byte oder bei einem Fehler den Wert -1 zurück.
Siehe auch: *Socket*, *recv* und *Send*.

SEND

Send sendet *AddrLen* Byte beginnend bei der Adresse *Addr* an den Socket *Sock*. Es muß eine Verbindung zum Socket bestehen. Die Funktion liefert die Anzahl der gesendeten Byte oder -1, falls ein Fehler bemerkt wurde. *Flags* kann folgende Werte annehmen:

1	Daten außerhalb des Bandes verarbeiten (Process out-of band data).
4	Routing umgehen und eine direkte Schnittstelle verwenden.

Fehler: Fehler werden im *SocketError* gespeichert. Es gibt folgende Fehler:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTCONN	Der Socket ist nicht verbunden.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	Die angegebene Adresse ist außerhalb der Adreßraums.
SYS_EMSGSIZE	Die Meldung kann nicht atomar (atomically) sein.
SYS_EWOULDBLOCK	Die angefragte Operation würde die Applikation blockieren.
SYS_ENOBUFS	Dem System steht nicht genug freier Pufferspeicher zur Verfügung.

Siehe auch: *recv*.

SendTo

```
function SendTo(Sock: LongInt; const Buf; BufLen: LongInt;
                Flags: LongInt; var Addr; AddrLen: LongInt): LongInt;
```

SendTo sendet Data aus dem Puffer *Buf* mit der Länge *BufLen* durch den Socket *Sock* mit den Optionen *Flags*. Die Daten werden an die Adresse *Addr* gesandt, die die Länge *AddrLen* hat.
Fehler: Bei einem Fehler wird der Wert -1 zurückgemeldet.
Siehe auch: *Socket*, *Send* und *RecvFrom*.

SetSocketOptions

```
function SetSocketOptions(Sock: LongInt; Level: LongInt;
                          OptName: LongInt;
                          const OptVal;
                          optlen: LongInt): LongInt;
```

SetSocketOptions legt die Verbindungsoptionen für den Socket *Sock* fest. Der Socket kann auf verschiedenen Levels geändert werden. Der Parameter *Level* bestimmt dies und kann folgende Werte annehmen:

SOL_SOCKET	Um den Socket selbst zu ändern.
XXX	Setzen Sie *Level* auf die Nummer des Protokolls, das die Optionen interpretieren soll. Weitere Informationen über diesen Befehl stehen in den Unix-Manpages unter *setsockopt*.

Fehler werden in *SocketError* gespeichert, wobei folgende Fehlercodes möglich sind:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.
SYS_EFAULT	*OptVal* verweist auf Speicher außerhalb des Adreßraums.

Siehe auch: *GetSocketOptions*

ShortHostToNet

```
function ShortHostToNet(Host: Word): Word;
```

ShortHostToNet wandelt eine Host-Portnummer in eine Netzwerk-Portnummer um. Dabei wird die Endianness der Hostmaschine berücksichtigt.
Siehe auch: *ShortNetToHost*, *HostToNet*, *NToHS* und *HToNS*.

ShortNetToHost

```
function ShortNetToHost(Net: Word): Word;
```

ShortNetToHost wandelt eine Netzwerk-Portnummer in eine Host-Portnummer um. Dabei wird auf die Endianness der Hostmaschine Rücksicht genommen.
Siehe auch: *ShortNetToHost*, *HostToNet*, *NToHS* und *HToNS*.

Shutdown(Sock: LongInt; How: LongInt): LongInt;

Shutdown beendet eine bidirektionale Verbindung, die durch *Sock* beschrieben wird. *How* legt fest, wie die Verbindung beendet wird:

0	Weiterer Empfang ist untersagt.
1	Weiteres Senden ist untersagt.
2	Weder Senden noch Empfangen sind erlaubt.

War die Funktion erfolgreich, wird 0, bei einem Fehler -1 zurückgeliefert.
Fehler: *SocketError* zur Fehlermeldung enthält das folgende:

SYS_EBADF	Der Socketdeskriptor ist ungültig.
SYS_ENOTCONN	Der Socket ist nicht verbunden.
SYS_ENOTSOCK	Der Deskriptor ist kein Socket.

Siehe auch: *Socket* und *Connect*

SOCK2FILE

```
procedure Sock2File(Sock: LongInt;
                    var SockIn: File;
                    var SockOut: File);
```

Sock2File wandelt den Socket *Sock* in zwei Pascal-Dateideskriptoren des Typs *File* um, einen zum Lesen aus dem Socket (*SockIn*) und einen für das Schreiben in den Socket (*SockOut*).
Siehe auch: *Socket* und *Sock2Text*.

SOCK2TEXT

```
procedure Sock2Text(Sock: LongInt; var SockIn: Text; var SockOut: Text);
```

Sock2Text wandelt den Socket *Sock* in zwei Pascal-Dateideskriptoren des Typs *Text* um, einen zum Lesen aus dem Socket (*SockIn*) und einen zum Schreiben in den Socket (*SockOut*).
Siehe auch: *Socket* und *Sock2File*.

SOCKET

```
function Socket(Domain: LongInt;
                SocketType: LongInt;
                Protocol: LongInt): LongInt;
```

Hinweis: Diese Funktion ist veraltet.
Socket erstellt einen neuen Socket in der Domain *Domain* vom Typ *SocketType* anhand des Protokolls *Protocol*. Die Domain, der Sockettyp und das Protokoll können anhand der vorgegebenen Konstanten (siehe Abschnitt zu den Konstanten, um geeignete Konstanten zu finden) festgelegt werden. War die Funktion erfolgreich, liefert sie einen Socketdeskriptor, der an einen nachfolgenden *Bind*-Befehl übermittelt werden kann. Falls die Funktion fehlgeschlagen ist, wird -1 zurückgeliefert.
Fehler: Fehler werden in *SocketError* gemeldet.
Für *Socket* gibt es folgende Fehlermeldungen:

SYS_EPROTONOSUPPORT	Der Protokolltyp oder das festgelegte Protokoll wird von dieser Domäne nicht unterstützt.
SYS_EMFILE	Die Deskriptortabelle dieses Prozesses ist voll.
SYS_ENFILE	Die Systemdateitabelle ist voll.
SYS_EACCESS	Es fehlen die Zugriffsrechte, um einen neuen Socket des festgelegten Typs und/oder Protokolls zu erstellen.
SYS_ENOBUFS	Nicht genügend Pufferspeicher. Der Socket kann nicht erstellt werden, bis genügend Ressourcen vorhanden sind.

Siehe auch: *SocketPair*, *Socket* (Unix-Handbuch)
Zum Beispiel siehe: *Accept*.
Siehe auch: *SocketPair*.

SocketError

```
function socketerror: cint;
```

SocketError enthält den Fehlercode der letzten Socket-Operation. Frägt den letzten Socket-Fehler ab.

SocketPair

```
function SocketPair(Domain: LongInt; SocketType: LongInt;
                    Protocol: LongInt; var Pair: TSockArray): LongInt;
```

Hinweis: Diese Funktion ist veraltet.
SocketPair erstellt zwei Sockets in der Domäne *Domain* vom Typ *SocketType* und verwendet das Protokoll *Protocol*. Die beiden werden in *Pair* zurückgegeben und sind unvereinbar. Die Funktion liefert -1 bei einem Fehler und 0 bei Erfolg zurück.
Fehler: Fehler werden in *SocketError* gemeldet und sind die gleichen wie in *Socket*
Siehe auch: *Str2UnixSockAddr* .

Str2UnixSockAddr

```
procedure Str2UnixSockAddr(const addr: String;
                           var t: TUnixSockAddr; var len: LongInt);
```

Str2UnixSockAddr ändert eine Unix-Socket-Adresse in einen String einer Struktur *TUnixSockAddr*, die an eine *Bind*-Anweisung übermittelt werden kann.
Siehe auch: *Socket* und *Bind*.

StrToHostAddr

```
function StrToHostAddr(IP: AnsiString): in_addr;
```

StrToHostAddr wandelt den String in *IP* in eine Hostadresse um und gibt diese zurück.
Fehler: Bei einem Fehler wird die Hostadresse mit Nullen gefüllt.
Siehe auch: *NetAddrToStr*, *HostAddrToStr* und *StrToNetAddr*.

StrToHostAddr6

```
function StrToHostAddr6(IP: String): Tin6_addr;
```

StrToHostAddr6 wandelt den String in *IP* in eine IPv6-Hostadresse um und gibt diese zurück.
Fehler: Bei einem Fehler wird die Hostadresse mit Nullen gefüllt.
Siehe auch: *NetAddrToStr6*, *HostAddrToStr6* und *StrToHostAddr*.

StrToNetAddr

```
function StrToNetAddr(IP: AnsiString): in_addr;
```

StrToNetAddr wandelt den String in *IP* in eine Netzwerkadresse um und gibt diese zurück.
Fehler: Bei einem Fehler wird die Netzwerkadresse mit Nullen gefüllt.
Siehe auch: *NetAddrToStr*, *HostAddrToStr* und *StrToHostAddr*

StrToNetAddr6

```
function StrToNetAddr6(IP: AnsiString): Tin6_addr;
```

StrToNetAddr wandelt den String in *IP* in eine IPv6-Netzwerkadresse um und gibt diese zurück.
Fehler: Bei einem Fehler wird die Netzwerkadresse mit Nullen gefüllt.
Siehe auch: *NetAddrToStr6*, *HostAddrToStr6* und *StrToHostAddr6*.

4.19 Unit ipc

Die Interprozeßkommunikations-Unit IPC von Free Pascal wurde von Michaël Van Canneyt für Linux geschrieben und und stellt die komplette Funktionalität der Interprozeßkommunikation von Unix System V zur Verfügung: Shared Memory (gemeinsam genutzter Speicher), Semaphoren (Signalübermittlung) und Messages (Nachrichten). Diese Unit funktioniert nur auf Linux.
Eine große Zahl von Konstanten sind in diesem Buch nur der Vollständigkeit halber aufgeführt und sollten unter normalen Umständen vom Programmierer nicht benötigt werden.
Die Unit *ipc* ruft die folgenden Units auf:

- BaseUnix,
- UnixType.

4.19.1 Konstanten, Typen und Variablen

Variablen

```
IPCerror: LongInt;
```

Die *IPCerror*-Variable wird für die Fehlermeldung aller Befehle herangezogen.

Konstanten

Diese Konstanten sind für das Erstellen von IPC-Nachrichten vorhanden:

IPC_CREAT	= 1 shl 9	Erstellt, wenn der Schlüssel nicht existiert.
IPC_EXCL	= 2 shl 9	Scheitert, wenn der Schlüssel existiert.
IPC_NOWAIT	= 4 shl 9	Fehler melden, wenn gewartet werden muß.

Diese Konstanten werden für die verschiedenartigen *xxxget*-Befehle benutzt:

IPC_RMID	= 0	Ressource entfernen.
IPC_SET	= 1	ipc_perm-Optionen setzen.
IPC_STAT	= 2	ipc_perm-Optionen auslesen.
IPC_INFO	= 3	IPCs anzeigen.

Diese Konstanten können an die verschiedenen *xxxctl*-Befehle übermittelt werden. Dabei handelt es sich um interne Steuercodes, die nicht verwendet werden sollten:

MSGMAX	= 4056	Interner Message-Steuercode. Nicht verwenden!
MSGMNB	= 16384	Interner Message-Steuercode. Nicht verwenden!
MSGMNI	= 128	Interner Message-Steuercode. Nicht verwenden!
MSG_EXCEPT	= 2 shl 12	Interner Message-Steuercode. Nicht verwenden!
MSG_NOERROR	= 1 shl 12	Interner Message-Steuercode. Nicht verwenden!

Konstanten für Semaphore:

SEM_GETALL	= 13	Semaphor-Operation: Alle Semaphorenwerte lesen.
SEM_GETNCNT	= 14	Semaphor-Operation: Zahl der Prozesse, die auf die Ressource warten, ermitteln.
SEM_GETPID	= 11	Semaphor-Operation: Prozeß-ID der letzten Operation ermitteln.

SEM_GETVAL	= 12	Semaphor-Operation: Aktuellen Wert der Semophore lesen.
SEM_GETZCNT	= 15	Semaphor-Operation: Zahl der Prozesse ermitteln, die darauf warten, daß die Semaphore den Wert 0 erreichen.
SEM_SEMMNI	= 128	Semaphor-Operation: ?
SEM_SEMMNS	=(SEM_SEMMNI * SEM_SEMMSL	Semaphor-Operation: ?
SEM_SEMMSL	= 32	Semaphor-Operation: ?
SEM_SEMOPM	= 32	Semaphor-Operation: ?
SEM_SEMVMX	= 32767	Semaphor-Operation: ?
SEM_SETALL	= 17	Semaphor-Operation: Alle Semaphorenwerte setzen.
SEM_SETVAL	= 16	Semaphor-Operation: Semaphorenwert setzen.
SEM_UNDO	= $1000	Konstante für *semop*.

Die folgenden Konstanten für den *shmctl*-Befehl sind nicht weiter beschrieben:

```
SHM_LOCK   = 11;
SHM_R      = 4 shl 6;
SHM_RDONLY = 1 shl 12;
SHM_REMAP  = 4 shl 12;
SHM_RND    = 2 shl 12;
SHM_UNLOCK = 12;
SHM_W      = 2 shl 6;
```

Typdeklarationen

```
TKey = cint;
```

TKey ist der durch die schlüsselerzeugende Funktion *ftok* zurückgegebene Typ.

```
key_t = TKey;
```

Alias für den Datentyp *TKey*.

```
msglen_t = culong;
```

Typ für die Längendefinition von Nachrichten.

```
msgqnum_t = culong;
```

Typ für die Numerierung von Message-Queues.

```
PSHMinfo = ^TSHMinfo;
TIPC_Perm = record
  key : TKey;
  uid : uid_t;
  gid : gid_t;
  cuid: uid_t;
  cgid: gid_t;
  mode: mode_t;
  seq : cushort;
end;
```

Die Struktur *TIPC_Perm* legt in allen IPC-Systemen Zugriffsrechte fest. Sie darf niemals direkt aufgerufen werden.

```
TMSG = record
  msg_next : PMSG;
  msg_type : LongInt;
  msg_spot : PChar;
  msg_stime: LongInt;
  msg_ts   : Integer;
end;
```

Der *TMSG*-Record ist für das Handling von Botschafts-Warteschlangen implementiert. Der Programmierer benötigt nie Zugriff auf diese Daten.

```
TMSGbuf = record
  mtype: clong;
  mtext: array[0..0] of Char;
end;
```

Der Record *TMSGbuf* enthält Daten über eine Botschaft. Dieser Record sollte niemals direkt angewandt werden, stattdessen besser ein eigener Record, der die Struktur des Records *TMSGbuf* aufweist und groß genug ist, um die Botschaft aufzunehmen. Das Feld *mtype* sollte immer vorhanden sein und einen gültigen Wert enthalten.

```
TMSGinfo = record
  msgpool: cint;
  msgmap : cint;
  msgmax : cint;
  msgmnb : cint;
  msgmni : cint;
  msgssz : cint;
  msgtql : cint;
  msgseg : cushort;
end;
```

Der Record *TMSGinfo* ist für das Handling von Botschafts-Warteschlangen implementiert und sollte nie direkt benutzt werden.

```
TMSQid_ds = record
  msg_perm  : TIPC_Perm;
  msg_first : PMSG;
  msg_last  : PMSG;
  msg_stime : time_t;
  msg_rtime : time_t;
  msg_ctime : time_t;
  msg_cbytes: Word;
  msg_qnum  : Word;
  msg_qbytes: Word;
  msg_lspid : ipc_pid_t;
  msg_lrpid : ipc_pid_t;
end;
```

Der Record *TMSQid_ds* wird von der Routine *msgctl* zurückgegeben und enthält alle Daten über die Botschafts-Warteschlange. Er sollte nie direkt aufgerufen werden, da es ein interner Kernel-Record ist, dessen Felder jederzeit geändert werden können.

```
TSEMbuf = record
  sem_num: cushort;
  sem_op : cshort;
  sem_flg: cshort;
end;
```

Der *TSEMbuf*-Record wird im *semop*-Befehl verwendet und gibt die Operationen an, die ausgeführt werden sollen.

```
TSEMid_ds = record
  sem_perm        : TIPC_Perm;
  sem_otime       : time_t;
  sem_ctime       : time_t;
  sem_base        : Pointer;
  sem_pending     : Pointer;
  sem_pending_last: Pointer;
  undo            : Pointer;
```

```
  sem_nsems       : cushort;
end;
```

Der Record *TSEMid_ds* wird vom Befehl *semctl* zurückgeliefert und enthält alle Daten, die sich auf die Signalübertragung (Semaphore) beziehen.

```
TSEMinfo = record
  semmap: cint;
  semmni: cint;
  semmns: cint;
  semmnu: cint;
  semmsl: cint;
  semopm: cint;
  semume: cint;
  semusz: cint;
  semvmx: cint;
  semaem: cint;
end;
```

Der *TSEMinfo*-Record wird vom Signalübertragungssystem verwendet und sollte nicht direkt benutzt werden.

```
PSEMun = ^TSEMun;
TSEMun = record
  case LongInt of
    0: (val   : LongInt);
    1: (buf   : PSEMid_ds);
    2: (arr   : PWord);
    3: (padbuf: PSeminfo);
    4: (padpad: Pointer);
end;
```

Der variante Record *TSEMun* (eigentlich eine C-Union) wird im *semctl*-Befehl verwendet.

```
TShmid_ds = record
  shm_perm  : TIPC_Perm;
  shm_segsz : cint;
  shm_atime : time_t;
  shm_dtime : time_t;
  shm_ctime : time_t;
  shm_cpid  : ipc_pid_t;
  shm_lpid  : ipc_pid_t;
  shm_nattch: Word;
  shm_npages: Word;
  shm_pages : Pointer;
  attaches  : Pointer;
end;
```

Die Struktur *TSHMid_ds* ermittelt oder setzt im Befehl *shmctl* die Einstellungen, die sich auf den gemeinsam genutzten Speicher beziehen.

```
TSHMinfo = record
  shmmax: cint;
  shmmin: cint;
  shmmni: cint;
  shmseg: cint;
  shmall: cint;
end;
```

Der *TSHMinfo*-Record wird intern im System des gemeinsam nutzbaren Speichers verwendet, der Programmierer darf nicht direkt auf einen solchen Record zugreifen.

Die Unit enthält außerdem eine ganze Reihe vorgefertigter Zeiger auf die zusammengesetzten Datentypen:

```
PIPC_Perm = ^TIPC_Perm;
```

Zeiger auf den Record *TIPC_Perm*.

```
PMSG = ^TMSG;
```

Zeiger auf den Record *TMSG*.

```
PMSGbuf = ^TMSGbuf;
```

Zeiger auf den Record *TMsgBuf* .

```
PMSGinfo = ^TMSGinfo;
```

Zeiger auf den Record *TMSGinfo*.

```
PMSQid_ds = ^TMSQid_ds;
```

Zeiger auf *TMSQid_ds*.

```
PSEMbuf = ^TSEMbuf;
```

Zeiger auf den Record *TSembuf*.

```
PSEMid_ds = ^TSEMid_ds;
```

Zeiger auf den Record *TSEMid_ds*.

```
PSEMinfo = ^TSEMinfo;
```

Zeiger auf den Record *TSEMinfo*.

```
PSEMun = ^TSEMun;
```

Zeiger auf den Record *TSEMun*.

```
PShmid_DS = ^TShmid_ds;
```

Zeiger auf den Record *TSHMid_ds*.

4.19.2 Prozeduren und Funktionen

FTOK

```
function ftok(Path: PChar; ID: cint): TKey;
```

ftok gibt einen Schlüssel zurück, der in einem *semget*-, *shmget*- oder *msgget*-Befehl auf eine neue oder bereits vorhandene IPC-Ressource zugreifen kann. *Path* ist der Name einer Datei im Dateisystem, *ID* ist ein beliebiges Zeichen. *ftok* bewirkt das gleiche wie sein Gegenstück in C; ein Pascalprogramm und ein C-Programm greifen auf die gleichen Ressourcen zu, falls sie den gleichen Pfad und die gleiche ID benutzen.
Fehler: *ftok* gibt -1 zurück, falls die Datei in *Path* nicht vorhanden ist.
Siehe auch: *semget*, *shmget* und *msgget*.
Beispiele für die Funktion sind bei *msgctl*, *semctl* und *shmctl* gezeigt.

MSGCTL

```
function msgctl(msqid: cint; cmd: cint; buf: PMSQid_ds): cint;
```

msgctl führt verschiedene Operationen in der Botschafts-Warteschlange mit der ID *msqid* aus. Die Operation hängt vom *cmd*-Parameter ab, der folgende Werte annehmen kann:

IPC_STAT	Die Funktion *msgctl* füllt den Record *TMSQid_ds* mit Informationen über die Botschafts-Warteschlange, wenn der *cmd*-Parameter *IPC_STAT* entspricht.
IPC_SET	Wenn *cmd* den Wert *IPC_SET* enthält, wird *buf* als Zeiger auf einen Record vom Typ *ipc_perm* interpretiert und setzt die Rechte der Warteschlange entsprechend dem Inhalt des Records.
IPC_RMID	Die Botschafts-Warteschlange wird aus dem System entfernt, wenn *IPC_RMDIR* als Wert des Parameters übergeben wird.

buf enthält die jeweils geforderten Daten. Wenn die Warteschlange entfernt werden soll, kann der Parameter NIL sein, da keine zusätzlichen Daten benötigt werden.
Die Funktion liefert *true*, falls erfolgreich, ansonsten *false*.
Fehler: Bei einem Fehler wird *false* zurückgegeben und *IPCerror* entsprechend gesetzt.
Siehe auch: *msgget*, *msgsnd* und *msgrcv*.

```
program msgtool;
uses
  ipc, baseunix;

type
  PMyMsgBuf = ^TMyMsgBuf;
  TMyMsgBuf = record
    mtype: LongInt;
    mtext: String[255];
  end;

procedure DoError(const Msg: String);
begin
  WriteLn(msg, ' returned an error: ', fpgeterrno);
  Halt(1);
end;

procedure SendMessage(Id: LongInt; var Buf: TMyMsgBuf;
                      MType: LongInt; const MText: String);
begin
  WriteLn('Sending message.');
  Buf.mtype := mtype;
  Buf.Mtext := mtext;
  if msgsnd(Id, PMsgBuf(@Buf), 256, 0) = -1 then
    DoError('msgsnd');
end;

procedure ReadMessage(ID: LongInt; var Buf: TMyMsgBuf; MType: LongInt);
begin
  WriteLn('Reading message.');
  Buf.MType := MType;
  if msgrcv(ID, PMSGBuf(@Buf), 256, mtype, 0) <> -1 then
    WriteLn('type: ', buf.mtype, ' Text: ', buf.mtext)
  else
    DoError('msgrcv');
end;

procedure RemoveQueue(ID: LongInt);
begin
  if msgctl(id, IPC_RMID, NIL) <> -1 then
    WriteLn('Removed Queue with id ', Id);
end;

procedure ChangeQueueMode(ID, mode: LongInt);
var
  QueueDS: TMSQid_ds;
begin
  if msgctl(Id, IPC_STAT, @QueueDS) = -1 then
    DoError('msgctl: stat');
  WriteLn('Old permissions: ', QueueDS.msg_perm.mode);
  QueueDS.msg_perm.mode := Mode;
```

```
  if msgctl(ID, IPC_SET, @QueueDS) = 0 then
    WriteLn('New permissions: ', QueueDS.msg_perm.mode)
  else
    DoError('msgctl: IPC_SET');
end;

procedure Usage;
begin
  WriteLn('Usage: msgtool s(end)    <type> <text> (max 255 characters)');
  WriteLn('               r(eceive) <type>');
  WriteLn('               d(elete)');
  WriteLn('               m(ode) <decimal mode>');
  Halt(1);
end;

function StrToInt(S: String): LongInt;
var
  M: LongInt;
  C: Integer;
begin
  Val(S, M, C);
  if C <> 0 then
    DoError('StrToInt: ' + S);
  StrToInt := M;
end;

var
  Key: TKey;
  ID : LongInt;
  Buf: TMyMsgBuf;
const
  ipckey = '.'#0;
begin
  if ParamCount < 1 then Usage;
  key := Ftok(@ipckey[1], Ord('M'));
  ID := msgget(key, IPC_CREAT or 438);
  if ID < 0 then DoError('MsgGet');
  case Upcase(Paramstr(1)[1]) of
    'S': if ParamCount <> 3 then Usage else
       SendMessage(id, Buf, StrToInt(Paramstr(2)), Paramstr(3));
    'R': if ParamCount <> 2 then Usage else
       ReadMessage(id, buf, StrToInt(Paramstr(2)));
    'D': if ParamCount <> 1 then Usage else
       RemoveQueue(ID);
    'M': if ParamCount <> 2 then Usage else
       ChangeQueueMode(id, StrToInt(Paramstr(2)));
    else Usage
  end;
end.
```

Listing 04.35: ipcex/msgtool.pp

MSGGET

```
function msgget(key: TKey; msgflg: cint): cint;
```

msgget gibt die ID der Meldungswarteschlange zurück, die von *TKey* beschrieben wird. Abhängig von den Flags in *msgflg* wird eine neue Warteschleife erzeugt.
msgflg kann einen oder mehrere der folgenden Werte besitzen (mit *OR* verknüpft):

IPC_CREAT	Die Warteschlange wird erzeugt, falls sie nicht schon angelegt ist.
IPC_EXCL	Wenn das Flag in Verbindung mit *IPC_CREAT* benutzt wird und die Warteschlange schon vorhanden ist, wird der Befehl abgebrochen. Das Flag kann nicht für sich allein stehen.

Optional können die Flags mit Zugriffsrechten über *OR* verknüpft werden. Die Zugriffsrechte kennen die gleichen Werte wie das Dateisystem.
Fehler: Bei einem Fehler wird -1 zurückgeliefert und *IPCError* wird gesetzt.
Siehe auch: *ftok*, *msgsnd*, *msgrcv*, *msgctl*, *semget*
Ein Beispiel ist bei *msgctl* gezeigt.

MSGRCV

```
function msgrcv(msqid: cint; msgp: PMSGbuf; msgsz: size_t;
                msgtyp: cint; msgflg: cint): cint;
```

msgrcv ruft eine Botschaft des Typs *msgtyp* aus der Botschafts-Warteschlange mit der ID *msqid* ab. *msgtyp* enthält den Typ, den die Botschaft besitzen soll, entspricht also dem *mtype*-Feld des *TMSGbuf*-Records. Die Botschaft wird in der durch *msgp* angegebenen *MSGbuf*-Struktur gespeichert.
Der Parameter *msgflg* kann zu Steuerung des Verhaltens des *msgrcv*-Befehls herangezogen werden. Der Parameter kann aus einer *OR*-Verknüpfung der folgenden Werte bestehen:

0	Keine besondere Bedeutung.
IPC_NOWAIT	Falls keine Meldungen verfügbar sind, erzeugt der Befehl sofort einen *ENOMSG*-Fehler.
MSG_NOERROR	Falls die Botschaft zu groß ist, wird kein Fehler erzeugt, sondern die Botschaft gekürzt. Normalerweise liefert der Befehl in solchen Fällen einen Fehler (*E2BIG*).

Die Funktion liefert *true*, wenn die Botschaft korrekt empfangen wurde, ansonsten *false*.
Fehler: Bei einem Fehler wird *false* zurückgegeben und *IPCerror* gesetzt.
Siehe auch: *msgget*, *msgsnd*, *msgctl*.
Zum Beispiel siehe *msgctl*.

MSGSND

```
function msgsnd(msqid: cint; msgp: PMSGbuf;
                msgsz: size_t;
                msgflg: cint): cint;
```

msgsnd sendet eine Botschaft mit der ID *msqid* an die Botschaftswarteschlange. *msgp* akzeptiert einen Zeiger auf einen Message-Record, der auf dem Typ *TMsgBuf* basieren sollte. *msgsiz* ist die Größe der Botschaft (nicht des gesamten Message-Records!).
Die Variable *msgflg* kann aus folgenden Werten zusammengesetzt sein (Verknüpfung mit *OR*):

0	Hat keine spezielle Bedeutung. Die Botschaft wird an die Warteschlange angehängt. Falls die Warteschlange voll ist, wird der aufrufende Prozeß blockiert, bis die Warteschlange wieder Botschaften aufnehmen kann.
IPC_NOWAIT	Falls die Warteschlange voll ist, wird die Botschaft verworfen und der Befehl kehrt sofort zurück.

Die Funktion liefert *true*, falls die Meldung erfolgreich gesendet wurde, ansonsten *false*.
Fehler: Bei einem Fehler liefert der Befehl *false* und *IPCerror* wird gesetzt.
Siehe auch: *msgget*, *msgrcv*, *seefmsgctl*
Ein Beispiel befindet sich unterhalb zur Erklärung zu *msgctl*.

SEMCTL

```
function semctl(semid: cint; semnum: cint; cmd: cint;
                var arg: TSEMun): cint;
```

semctl führt verschiedene Operationen auf die Nachricht *semnum* im Signalsystem mit der ID *semid* aus.
Der Parameter *arg* wird je nach auszuführender Operation verschieden interpretiert. Es gibt folgende Darstellungen des varianten Records:

```
type
  TSEMun = record
    case LongInt of
      0: (val   : LongInt);
      1: (buf   : PSEMid_ds);
      2: (arr   : PWord);
      3: (padbuf: PSeminfo);
      4: (padpad: Pointer);
    end;
```

Welche Operation ausgeführt wird, hängt vom Parameter *cmd* ab, der einen der folgenden Werte annehmen kann:

IPC_STAT	In diesem Fall sollte das Feld *buf* des Parameters *arg* gesetzt sein und die Adresse eines *TSEMid_ds*-Records enthalten. Der Befehl *semctl* füllt diese Struktur *TSEMid_ds* mit Informationen über das Signalsystem.
IPC_SET	In diesem Fall sollte wiederum das Feld *buf* von *arg* die Adresse eines *TSEMid_ds*-Records enthalten. Die Zugriffsrechte des Signalsystems werden gemäß den Angaben im *ipc_perm*-Record festgelegt.
IPC_RMID	Mit diesem Kommando wird das Signalsystem entfernt.
GETALL	In diesem Fall sollte das Feld *arr* von *arg* auf einen Speicherbereich verweisen, in dem die Werte der Signale gespeichert werden. Die Größe dieses Speicherbereiches ist »*SizeOf(Word)* * Anzahl der Signale im System«. Dieser Aufruf füllt dann das Array mit den Werten der Signale.
GETNCNT	Das Feld *val* des varianten Records ist bei diesem Befehl gültig und enthält nach dem Aufruf die Anzahl der Prozesse, die auf Ressourcen warten.
GETPID	*semctl* gibt die Prozeß-ID des Prozesses zurück, der den letzten *semop*-Befehl ausführte.
GETVAL	*semctl* gibt den Wert des Signals mit der Nummer *semnum* zurück.
GETZCNT	*semctl* gibt die Anzahl der Prozesse zurück, die darauf warten, daß Signale den Wert Null erreichen.
SETALL	In diesem Fall sollte das Feld *arr* von *arg* auf einen Speicherbereich verweisen, in dem die neuen Werte der Signale gespeichert sind. Diese Werte überschreiben bei Angabe von *SETALL* dann die bisherigen Einstellungen. Die Größe des Speicherbereichs kann zu »*SizeOf(Word)* * Anzahl der Signale im System« ermittelt werden.
SETVAL	Der Wert des Signals Nummer *semnum* wird auf den Wert des Feldes *val* des Parameters *arg* gesetzt.

Fehler: Bei einem Fehler liefert die Funktion -1 und *IPCerror* wird entsprechend gesetzt.
Siehe auch: *semget* und *semop*.

```
program semtool;
(* program to demonstrate the use of semaphores *)

uses
  ipc, baseunix;
const
  MaxSemValue = 5;

procedure DoError(const Msg: String);
begin
  WriteLn('Error: ',msg,' Code: ', fpgeterrno);
  Halt(1);
end;

function getsemval(ID, Member: LongInt): LongInt;
var
  S: TSEMun;
begin
  GetSemVal := SemCtl(id, Member, SEM_GETVAL, S);
end;

procedure DispVal(ID, Member: LongInt);
begin
  WriteLn('Value for Member ', Member, ' is ',
          GetSemVal(ID, Member));
end;

function GetMemberCount(ID: LongInt): LongInt;
var
  opts : TSEMun;
  semds: TSEMid_ds;
begin
  opts.buf := @semds;
  if semctl(Id, 0, IPC_STAT,opts) <> -1 then
    GetMemberCount := semds.sem_nsems else GetMemberCount := -1;
end;

function OpenSem(Key: TKey): LongInt;
begin
  OpenSem := semget(Key, 0, 438);
  if OpenSem = -1 then DoError('OpenSem');
end;

function CreateSem(Key: TKey; Members: LongInt): LongInt;
var
  Count  : LongInt;
  Semopts: TSemun;
begin
  // the semmsl constant seems kernel specific
  if Members > semmsl then
    DoError('Sorry, maximum number of semaphores in set exceeded');
  WriteLn('Trying to create a new semaphore set with ',
          Members, ' Members.');
```

```
  CreateSem := semget(key, Members, IPC_CREAT or IPC_Excl or 438);
  if CreateSem = -1 then
    DoError('Semaphore set already exists.');
  Semopts.Val := MaxSemValue;
  // Initial value of semaphores
  for Count := 0 to Members - 1 do
    semctl(CreateSem,count,SEM_SETVAL,semopts);
end;

procedure lockSem(ID, Member: LongInt);
var
  lock: TSEMbuf;
begin
  with lock do begin
    sem_num := 0;
    sem_op  := -1;
    sem_flg := IPC_NOWAIT;
  end;
  if (Member < 0) or (Member > GetMemberCount(ID) - 1) then
    DoError('semaphore Member out of range');
  if getsemval(ID, Member) = 0 then
    DoError('Semaphore resources exhausted (no lock)');
  lock.sem_num := Member;
  WriteLn('Attempting to lock Member ', Member,
          ' of semaphore ', ID);
  if semop(Id, @lock, 1) = -1 then
    DoError('Lock failed')
  else
    WriteLn('Semaphore resources decremented by one');
  dispval(ID,Member);
end;

procedure UnlockSem(ID, Member: LongInt);
var
  Unlock: TSEMbuf;
begin
  with Unlock do begin
    sem_num := 0;
    sem_op  := 1;
    sem_flg := IPC_NOWAIT;
  end;
  if (Member < 0) or (Member > GetMemberCount(ID) - 1) then
    DoError('semaphore Member out of range');
  if getsemval(ID,Member) = MaxSemValue then
     DoError('Semaphore not locked');
  Unlock.sem_num := Member;
  WriteLn('Attempting to unlock Member ',
          Member, ' of semaphore ', ID);
  if semop(Id, @unlock, 1) = -1 then
    DoError('Unlock failed')
 else
    WriteLn('Semaphore resources incremented by one');
  dispval(ID,Member);
end;

procedure RemoveSem(ID: LongInt);
var
```

```
  S: TSemun;
begin
  if semctl(Id, 0, IPC_RMID, s) <> -1 then
    WriteLn('Semaphore removed')
  else
    DoError('Couldn''t remove semaphore');
end;

procedure ChangeMode(ID, Mode: LongInt);
var
  rc   : LongInt;
  opts : TSEMun;
  semds: TSEMid_ds;
begin
  opts.buf := @semds;
  if not semctl(Id, 0, IPC_STAT, opts) <> -1 then
    DoError('Couldn''t stat semaphore');
  WriteLn('Old permissions were: ', semds.sem_perm.mode);
  semds.sem_perm.mode := mode;
  if semctl(id, 0, IPC_SET, opts) <> -1 then
    WriteLn('Set permissions to ', mode)
  else
    DoError('Couldn''t set permissions');
end;

procedure PrintSem(ID: LongInt);
var
  I, cnt: LongInt;
begin
  cnt := getMembercount(ID);
  WriteLn('Semaphore ', ID,' has ', cnt, ' Members');
  for I := 0 to cnt - 1 Do  DispVal(id,i);
end;

procedure Usage;
begin
  WriteLn('Usage: semtool c(reate) <count>');
  WriteLn('               l(ock) <Member>');
  WriteLn('               u(nlock) <Member>');
  WriteLn('               d(elete)');
  WriteLn('               m(ode) <mode>');
  Halt(1);
end;

function StrToInt(S: String): LongInt;
var
  M: LongInt;
  C: Integer;
begin
  Val(S, M, C);
  if C <> 0 then DoError('StrToInt: ' + S);
  StrToInt := M;
end;

var
  Key: TKey;
  ID : LongInt;
```

```
const
  ipckey='.'#0;
begin
  if ParamCount < 1 then Usage;
  key := ftok(@ipckey[1], Ord('s'));
  Case UpCase(Paramstr(1)[1]) of
    'C': begin
      if ParamCount <> 2 then Usage;
      CreateSem(key, StrToInt(ParamStr(2)));
    end;
    'L': begin
    if ParamCount <> 2 then Usage;
      ID := OpenSem(key);
      LockSem(ID, StrToInt(ParamStr(2)));
    end;
   'U': begin
     if ParamCount <> 2 then Usage;
     ID := OpenSem(key);
     UnLockSem(ID, StrToInt(ParamStr(2)));
   end;
   'M': begin
     if ParamCount <> 2 then Usage;
     ID := OpenSem(key);
     ChangeMode(ID, StrToInt(ParamStr(2)));
   end;
   'D': Begin
     ID := OpenSem(Key);
     RemoveSem(Id);
   end;
   'P': begin
     ID := OpenSem(Key);
     PrintSem(Id);
   end;
  else Usage
  end;
end.
```

Listing 04.36: ipcex/semtool.pp

SEMGET

```
function semget(key: TKey; nsems: cint; semflg: cint): cint;
```

semget gibt die ID des durch *key* beschriebenen Signalsystems zurück. In Abhängigkeit vom Inhalt des Parameters *semfls* können die folgenden Aktionen durchgeführt werden (Verknüpfung über *OR* möglich):

IPC_CREAT	Das System wird angelegt, falls es noch nicht vorhanden ist.
IPC_EXCL	Dieses Flag kann nur zusätzlich zur *IPC_CREAT*-Option angegeben werden und bewirkt einen Fehler, wenn das Signalsystem bereits vorhanden ist.

Die Flags können mit Zugriffsrechten über *OR* verknüpft werden, die die gleichen Werte wie ihre Pendants im Dateisystem haben.
Falls ein neues Signalsystem erzeugt wurde, kann dieses *nsems* Signale aufnehmen.
Fehler: Bei einem Fehler wird -1 zurückgegeben und *IPCError* wird gesetzt.
Siehe auch: *ftok*, *semop* und *semctl*.

SEMOP

```
function semop(semid: cint; sops: PSEMbuf; nsops: cuint): cint;
```

semop führt eine Reihe von Operationen auf verschiedene Signale durch. *sops* zeigt auf ein Array vom Typ *TSEMbuf*. Das Array sollte *nsops* Elemente enthalten.
Das Feld einer *TSEMbuf*-Struktur

```
TSEMbuf = record
  sem_num : Word;
  sem_op  : Integer;
  sem_flg : Integer;
end;
```

sollte mit folgenden Werten gefüllt sein:

sem_num	Die Nummer des Signals im System, auf die die Operation durchgeführt werden soll.
sem_op	Die auszuführende Operation. Sie kann folgende Werte annehmen: 1 Eine positive Zahl wird zum aktuellen Wert des Signals addiert. 2 Falls 0 (Null) angegeben wurde, wird der Prozeß angehalten, bis das spezifizierte Signal den Wert Null erreicht hat. 3 Falls eine negative Zahl angegeben ist, wird sie vom aktuellen Wert des Signals subtrahiert. Wenn der Wert des Signals dabei negativ wird, wird der Prozeß angehalten, bis der Wert wieder groß genug wird. Mit Angabe des *IPC_NOWAIT*-Flags in *sem_flg* kann dieses Verhalten umgangen werden.
sem_flg	Falls *IPC_NOWAIT* angegeben wurde, wird der aufrufende Prozeß nie angehalten.

Die Funktion liefert *True*, falls die Operationen erfolgreich waren, ansonsten *False*.
Fehler: Bei einem Fehler wird *False* zurückgeliefert und *IPCerror* wird gesetzt.
Siehe auch: *semget* und *semctl*.

SHMAT

```
function shmat(ShmID: cint; shmaddr: Pointer; shmflg: cint): Pointer;
```

shmat bindet einen gemeinsam nutzbaren Speicherblock mit der ID *shmid* an den aktuellen Prozeß und gibt einen Zeiger auf diesen zurück. Falls *shmaddr* gleich NIL ist, wählt das System einen bisher nicht zugeteilten Speicherbereich aus, der sich so hoch wie möglich im Adreßraum befindet. Falls *shmaddr* verschieden von NIL ist und *SHM_RND* in *shmflg* angegeben ist, wird *shmaddr* auf *SHMLBA* abgerundet. Falls *SHM_RND* nicht angegeben wurde, muß *shmaddr* an Seitengrenzen ausgerichtet sein.
Der Parameter *shmflg* kann das Verhaltens von *shmat* steuern und folgende Werte annehmen (Verknüpfung mit *OR*):

SHM_RND	Die angegebene Adresse in *shmaddr* wird zu *SHMLBA* abgerundet.
SHM_RDONLY	Es besteht nur Lesezugriff auf den Speicherblock, wenn diese Option angegeben wurde, ansonsten wird der Speicher für Lesen und Schreiben hinzugefügt. Die Applikation braucht Lese- und Schreibzugriff, um den geteilten Speicher zu verwalten.

Fehler: Bei einem Fehler wird -1 zurückgeliefert und *IPCError* wird gesetzt.
Siehe auch: *shmget*, *shmdt*, *shmctl*.
Zum Beispiel siehe *shmctl*.

SHMCTL

```
function ShmCtl(ShmID: cint; cmd: cint; buf: PShmid_DS): cint;
```

shmctl führt verschiedene Operationen auf den gemeinsam nutzbaren Speicherblock aus, der durch die ID *shmid* identifiziert wird.
Der Parameter *buf* zeigt auf einen *TSHMid_ds*-Record, der für die Operation verwendet werden soll. Der Parameter *cmd* bestimmt, welche Operation ausgeführt werden soll. Folgende Werte sind möglich:

IPC_STAT	*shmctl* füllt den *TSHMid_ds*-Record, auf den *buf* verweist, mit Informationen über den gemeinsam genutzten Speicherblock.
IPC_SET	*shmctl* wendet die Werte im *ipc_perm*-Record auf den Speicherblock an.
IPC_RMID	*shmctl* entfernt den Speicherblock aus dem System (nachdem alle Prozesse sich von ihm abgekoppelt haben). Falls erfolgreich, liefert die Funktion *True*, ansonsten *False*.

Fehler: Bei einem Fehler wird *false* zurückgeliefert und *IPCError* wird gesetzt.
Siehe auch: *shmget*, *shmat* und *shmdt*.

```
program shmtool;
uses
  ipc, Strings, Baseunix;
const
  SegSize = 100;
var
  key        : TKey;
  ShmID, cntr: LongInt;
  SegPtr     : PChar;

procedure Usage;
begin
  WriteLn('Usage: shmtool w(rite) text');
  WriteLn('               r(ead)');
  WriteLn('               d(elete)');
  WriteLn('               m(ode change) mode');
  Halt(1);
end;

procedure Writeshm(ID: LongInt; ptr: PChar; S: String);
begin
  StrCopy(ptr, s);
end;

procedure ReadSHM(ID: LongInt; ptr: PChar);
begin
  WriteLn('Read: ',ptr);
end;

procedure RemoveSHM(ID: LongInt);
begin
  ShmCtl(ID, IPC_RMID, NIL);
  WriteLn('Shared memory marked for deletion');
end;

procedure ChangeMode(ID: LongInt; mode: String);
var
```

```
  m   : Word;
  code: Integer;
  data: TSHMid_ds;
begin
  Val(mode, m, code);
  if code <> 0 then Usage;
  if ShmCtl(ShmID, IPC_STAT, @data) = -1 then begin
    WriteLn('Error: ShmCtl: ', fpgeterrno);
    Halt(1);
  end;
  WriteLn('Old permissions: ', data.shm_perm.mode);
  data.shm_perm.mode := m;
  if ShmCtl(ShmID, IPC_SET, @data) = -1 then begin
    WriteLn('Error: ShmCtl :', fpgeterrno);
    Halt(1);
  end;
  WriteLn('New permissions: ', data.shm_perm.mode);
end;

const
  ftokpath = '.'#0;
begin
  if ParamCount < 1 then Usage;
   key := ftok(PChar(@ftokpath[1]), Ord('S'));
  ShmID := shmget(key, segsize, IPC_CREAT or IPC_EXCL or 438);
  if ShmID = -1 then begin
    WriteLn('Shared memory exists. Opening as client');
    ShmID := shmget(key, segsize, 0);
    if ShmID = -1 then begin
      WriteLn('shmget: Error!', fpgeterrno);
      Halt(1);
    end
  end else
    WriteLn('Creating new shared memory segment.');
  SegPtr := shmat(ShmID, NIL, 0);
  if LongInt(SegPtr) = -1 then begin
    WriteLn('Shmat: error!', fpgeterrno);
    Halt(1);
  end;
  case Upcase(ParamStr(1)[1]) of
    'W': writeshm(ShmID, SegPtr, ParamStr(2));
    'R': ReadSHM(ShmID, SegPtr);
    'D': RemoveSHM(ShmID);
    'M': ChangeMode(ShmID, ParamStr(2));
    else
      WriteLn(ParamStr(1));
      Usage;
  end;
end.
```

Listing 04.37: ipcex/shmtool.pp

SHMDT

```
function shmdt(shmaddr: Pointer): cint;
```

shmdt trennt den Speicher der Adresse *shmaddr* vom Prozeß ab. Dieser Speicherblock ist für den aktuellen Prozeß nicht mehr zugänglich, bis er durch einen *shmat*-Befehl wieder angefügt wird.

Die Funktion liefert *true*, falls der Speicherblock erfolgreich abgetrennt wurde, ansonsten *False*.
Fehler: Bei einem Fehler wird *False* zurückgeliefert und *IPCError* wird gesetzt.
Siehe auch: *shmget*, *shmat*, *shmctl*

SHMGET

```
function shmget(key: TKey; size: size_t; flag: cint): cint;
```

shmget liefert die ID des gemeinsam nutzbaren Speicherblocks, der durch *key* beschrieben wird. Abhängig von den Flags in *flag* wird ein neuer Speicherblock erstellt.
flag kann einen oder mehreren der folgenden Werte annehmen (Kombination mit *OR*):

IPC_CREAT	Die Warteschlange wird angelegt, falls sie nicht schon vorhanden ist.
IPC_EXCL	Kann nur in Kombination mit *IPC_CREAT* angegeben werden und führt zu einem Fehler, falls der Speicherblock bereits vorhanden ist.

Optional können die Flags noch mit Zugriffsrechten verknüpft werden, deren Werte den Zugriffsrechten von Dateien entsprechen.
Wird ein neuer Speicherblock erstellt, hat er die Größe *Size*.
Fehler: Bei einem Fehler wird -1 zurückgegeben und *IPCError* gesetzt.
Siehe auch: *shmat*, *shmdt*, *shmctl*

4.20 Unit Video

Die Unit *Video* stellte eine systemunabhängige Zwischenschicht für den Zugriff auf den Bildschirm zur Verfügung, weshalb auf allen Plattformen, für die die Unit zur Verfügung gestellt ist, einheitlich auf den Textbildschirm geschrieben werden kann.
Die Arbeitsweise der Bildschirmroutinen ist sehr einfach. Nach dem Aufruf von *InitVideo* enthält das Array *VideoBuf* eine Zwischenspeicherung des Bildschirms der Größe *ScreenWidth*ScreenHeight* und zwar von links nach recht und oben nach unten, wenn man die Elemente des Arrays durchläuft. *VideoBuf[0]* enthält damit das Zeichen und den Farbcode der obersten linken Ecke des Bildschirms und *VideoBuf[ScreenWidth]* die Daten für die Zeichen in der ersten Spalte in der zweiten Reihe auf dem Bildschirm und so weiter.
Um in diesen »Bildschirm« zu schreiben, muß der Text in das Array *VideoBuf* geschrieben werden. Der Aufruf von *UpdateScreen* kopiert dann den Text in möglichst idealer Form in den Bildschirm, was in diesem Kapitel auch noch an einem Beispiel gezeigt wird.
Das Attribut für die Farbe ist eine Kombination aus Vordergrund- und Hintergrundfarbe, ergänzt um das Blinkbit. Die folgenden Bitangaben beschreiben die diversen Farbkombinationen:

Bit 0-3	Die Vordergrundfarbe, hierfür können alle Farbkonstanten eingesetzt werden.
Bit 4-6	Die Hintergrundfarbe, sie kann auf einen Teilbereich der dunklen Farben der Farbkonstanten gesetzt werden.
Bit 7	Das Blinkbit. Ist es gesetzt, blinkt das Zeichen auf dem Bildschirm.

Jede mögliche Farbe hat eine zugeordnete Konstante, die alle in der Liste der Konstanten ab Seite O80 aufgeführt sind. Die Vordergrund- und die Hintergrundfarbe können auch direkt zu einem Farbattribut verschmolzen werden:

```
Attr := ForeGroundColor + (BackGroundColor shl 4);
```

Außerdem kann das Farbattribut auch über ein logisches OR mit dem Blinkattribut zusammengefügt werden, um ein blinkendes Zeichen zu erreichen:

```
Attr := Attr or blink;
```

Es darf aber nicht unerwähnt bleiben, daß diese Optionen nicht von allen Treibern unterstützt werden.
Der Inhalt des Arrays *VideoBuf* kann verändert werden. Das entspricht einem »Schreiben« auf den Bildschirm. Sind alle Daten in diesen virtuellen Bildschirm geschrieben, wird der Bildschirmpuffer wieder mit dem bereits erwähnten *UpdateScreen* in den wirklichen Bildschirmspeicher übertragen.
Die Aktualisierung des Bildschirms kann, um die Arbeit zu beschleunigen, verboten werden. Dafür ist die Funktion *LockScreenUpdate* vorgesehen, die einen internen Zähler erhöht. Solange er größer als 0 ist, bewirkt der Aufruf von *UpdateScreen* nichts. Der Zähler wird mit *UnlockScreenUpdate* wieder verringert. Hat er den Wert 0 erreicht, aktualisiert der nächste Aufruf von *UpdateScreen* die Bildschirmdaten. Dies ist vor allem dann sinnvoll, wenn verschachtelte Funktionen eine große Zahl von Bildschirmausgaben tätigen.
Die Unit *Video* besitzt auch eine Schnittstelle, um eigene Bildschirmtreiber zu schreiben. Damit ist es möglich, den voreingestellten Bildschirmtreiber mit einem eigenen zu überschreiben. Wie das geht, ist bei der Funktion *SetVideoDriver* erläutert. Der aktuelle Bildschirmtreiber kann mit *GetVideoDriver* ermittelt werden.

Hinweis: Die Unit *Video* sollte nicht gleichzeitig mit der veralteten an Turbo Pascal angelehnten Unit *Crt* verwendet werden. Das führt zu recht seltsamem Programm- und Bildschirmverhalten und Abstürze sind wahrscheinlich.
Die Beispiele in diesem Abschnitt rufen außer der Unit *Video* auch die Unit *VidUtil* auf, die als einzige Routine die Prozedur *TextOut* enthält. Sie schreibt einen Text an der angegebenen Stelle auf den Bildschirm und sieht folgendermaßen aus:

```
unit vidutil;

Interface

uses
  video;

procedure TextOut(X, Y: Word; const S: String);

implementation

procedure TextOut(X, Y: Word; const S : String);
var
  W, P, I, M: Word;
begin
  P := ((X-1) + (Y-1) * ScreenWidth);
  M := Length(S);
  if P + M > ScreenWidth * ScreenHeight then
    M := ScreenWidth*ScreenHeight - P;
  for I := 1 to M do
    VideoBuf^[P + I - 1] := Ord(S[i]) + ($07 shl 8);
end;
end.
```

4.20.1 Schreiben eines eigenen Bildschirmtreibers

Das Schreiben eines eigenen Bildschirmtreibers ist gar nicht kompliziert und bedeutet eigentlich nur, daß eine Reihe von Funktionen, die mit *SetVideoDriver* registriert werden, geschrieben werden müssen. Die verschiedenen zur Implementierung zur Verfügung stehenden Funktionen sind im Record *TVideoDriver* enthalten:

```
TVideoDriver = record
  InitDriver       : procedure;
  DoneDriver       : procedure;
  UpdateScreen     : procedure(Force: Boolean);
  ClearScreen      : procedure;
  SetVideoMode     : function(const Mode: TVideoMode): Boolean;
  GetVideoModeCount: function: Word;
  GetVideoModeData : function(Index : Word; var Data: TVideoMode): Boolean;
  SetCursorPos     : procedure(NewCursorX, NewCursorY: Word);
  GetCursorType    : function: Word;
  SetCursorType    : procedure(NewType: Word);
  GetCapabilities  : function: Word;
end;
```

Es müssen allerdings nicht alle diese Funktionen neu implementiert werden, notwendig ist nur *UpdateScreen*. Die allgemeinen Funktionen in der Unit *Video* prüfen, welche Funktionalität der Treiber zur Verfügung stellt. Die Funktionalität dieser Aufrufe ist die selbe wie bei den in diesem Kapitel beschriebenen Aufrufen der Unit *Video*, womit das erwartete Verhalten aus den Beschreibungen in diesem Abschnitt erschlossen werden kann. Einige der Aufrufe benötigen allerdings ein paar zusätzliche Hinweise:

InitDriver	Diese Prozedur wird von *InitVideo* aufgerufen. Sie sollte alle Datenstrukturen, die für die Funktionsfähigkeit des Treibers benötigt werden, bereitstellen und möglicherweise auch die Bildschirminitialisierungen. Es muß dafür gesorgt sein, daß die Prozedur garantiert nur einmal aufgerufen wird, ein erneuter Aufruf darf nur nach einem *DoneVideo* erfolgen. Die Variablen *ScreenWidth* und *ScreenHeight* sollten nach einem Aufruf dieser Prozedur richtig initialisiert sein, da *InitVideo* die beiden Bildschirmpuffer *VideoBuf* und *OldVideoBuf* auf den Werten dieser Variablen initialisiert.
DoneDriver	Diese Prozedur sollte alle von *InitDriver* aufgebauten Strukturen wieder freigeben. Außerdem sollte möglichst auch der Bildschirm auf die Werte vor der Initialisierung des Treibers zurückgesetzt werden. Die beiden Arrays *VideoBuf* und *OldVideoBuf* werden vom allgemeinen Aufruf *DoneVideo* wieder freigegeben.
UpdateScreen	Die einzige unbedingt notwendige Funktion des Treibers. Sie sollte den Bildschirm auf Basis des Inhalts des Arrays *VideoBuf* aktualisieren. Dieser Vorgang wird optimiert, indem die Werte gegen die des Arrays *OldVideoBuf* abgeglichen werden. Nach dem Aktualisieren des Bildschirms muß die Prozedur *UpdateScreen* den Puffer *OldVideoBuf* eigenständig aktualisieren. Ist der Parameter *Force* auf *True* gesetzt, wird der gesamte Bildschirm aktualisiert, nicht nur die geänderten Werte.
ClearScreen	Wenn es eine schnellere Möglichkeit gibt, den Bildschirm zu löschen, als alle Zeichenzellen mit Leerzeichen zu überschreiben, kann das an dieser Stelle eingebaut werden. Ist diese Funktion im Treiber nicht implementiert, schreibt die allgemeine Routine Leerzeichen in alle Bildschirmzellen und ruft anschließend die Prozedur *UpdateScreen(True)* auf.
SetVideoMode	Diese Funktion setzt, falls sie verfügbar ist, den gewünschten Bildschirmmodus. Sie sollte *True* zurückgeben, wenn der Bildschirmmodus gesetzt werden konnte und *False*, wenn nicht.
GetVideoModeCount	Sollte die Zahl der unterstützten Bildschirmmodi melden. Falls keine unterschiedlichen Bildschirmmodi erkannt werden, sollte diese Funktion nicht implementiert werden, die allgemeine Routine gibt 1 zurück (was für den aktuellen Modus steht).
GetVideoModeData	Diese Funktion sollte die Daten für den Modus mit dem angegebenen *Index* melden. *Index* ist nullbasiert. Die Funktion sollte *True* zurückgeben, wenn die Daten richtig gemeldet wurden, und *False*, wenn in *Index* ein ungültiger Wert angegeben wurde. Ist diese Funktion nicht implementiert, gibt die allgemeine Routine die Daten des aktuellen Bildschirmmodus zurück, wenn für *Index* der Wert 0 angegeben ist.
GetCapabilities	Ist diese Funktion nicht implementiert, wird von der allgemeinen Funktion 0 (das heißt, keine Fähigkeiten) zurückgegeben.

Die folgende Unit zeigt, wie ein Bildschirmtreiber mit einem Treiber, der Debuginformationen in eine Datei ausgibt, überschrieben wird. Die Unit kann in Beispielprogramme eingebunden werden, indem sie einfach der *Uses*-Klausel hinzugefügt wird. Das Setzen von *DetailedVideoLogging* auf *True* erzeugt ein detailliertes Protokoll, bremst dabei aber das Programm stark ab.

```
unit viddbg;

interface

uses
  video;

procedure StartVideoLogging;
procedure StopVideoLogging;
function  IsVideoLogging: Boolean;
procedure SetVideoLogFileName(FileName: String);

const
  DetailedVideoLogging: Boolean = False;

implementation

uses
  sysutils, keyboard;

var
  NewVideoDriver,
  OldVideoDriver : TVideoDriver;
  Active,Logging : Boolean;
  LogFileName    : String;
  VideoLog       : Text;

function TimeStamp: String;
begin
  TimeStamp := FormatDateTime('hh:nn:ss', Time());
end;

procedure StartVideoLogging;
begin
  Logging := true;
  WriteLn(VideoLog, 'Start logging video operations at: ',
          TimeStamp);
end;

procedure StopVideoLogging;
begin
  WriteLn(VideoLog, 'Stop logging video operations at: ',
          TimeStamp);
  Logging := false;
end;

function IsVideoLogging: Boolean;
begin
  IsVideoLogging := Logging;
end;

var
  ColUpd, RowUpd: array[0..1024] of Integer;

procedure DumpScreenStatistics(Force: Boolean);
var
  I, Count: Integer;
```

```
begin
  if Force then Write(VideoLog, 'forced ');
  WriteLn(VideoLog, 'video update at ', TimeStamp, ' : ');
  FillChar(Colupd, SizeOf(ColUpd), #0);
  FillChar(Rowupd, SizeOf(RowUpd), #0);
  Count := 0;
  for I := 0 to VideoBufSize div SizeOf(TVideoCell) do begin
    if VideoBuf^[i] <> OldVideoBuf^[i] then begin
      Inc(Count);
      Inc(ColUpd[I mod ScreenWidth]);
      Inc(RowUpd[I div ScreenHeight]);
    end;
  end;
  Write(VideoLog, Count, ' videocells differed divided over ');
  Count := 0;
  for I := 0 to ScreenWidth - 1 do
    if ColUpd[I] <> 0 then Inc(Count);
  Write(VideoLog, Count, ' columns and ');
  Count := 0;
  for I := 0 to ScreenHeight - 1 do
    if RowUpd[I] <> 0 then Inc(Count);
  WriteLn(VideoLog,Count,' rows.');
  if DetailedVideoLogging then begin
    for I:=0 to ScreenWidth - 1 do
      if ColUpd[I] <> 0 then
        Writeln(VideoLog, 'Col ', i, ' : ', ColUpd[I]:3,
                ' rows changed');
    for I := 0 to ScreenHeight - 1 do
      if RowUpd[I] <> 0 then
        WriteLn(VideoLog, 'Row ', i, ' : ', RowUpd[I]:3,
                ' colums changed');
  end;
end;

procedure LogUpdateScreen(Force: Boolean);
begin
  if Logging then DumpScreenStatistics(Force);
  OldVideoDriver.UpdateScreen(Force);
end;

procedure LogInitVideo;
begin
  OldVideoDriver.InitDriver;
  Assign(VideoLog, logFileName);
  Rewrite(VideoLog);
  Active := true;
  StartVideoLogging;
end;

procedure LogDoneVideo;
begin
  StopVideoLogging;
  Close(VideoLog);
  Active := false;
  OldVideoDriver.DoneDriver;
end;
```

```
procedure SetVideoLogFileName(FileName: String);
begin
  if not Active then LogFileName := FileName;
end;

initialization
  GetVideoDriver(OldVideoDriver);
  NewVideoDriver := OldVideoDriver;
  NewVideoDriver.UpdateScreen := @LogUpdateScreen;
  NewVideoDriver.InitDriver   := @LogInitVideo;
  NewVideoDriver.DoneDriver   := @LogDoneVideo;
  LogFileName := 'Video.log';
  Logging     := false;
  SetVideoDriver(NewVideoDriver);
end.
```

4.20.2 Konstanten, Typen und Variablen

Konstanten

Die folgenden Angaben stehen für die Farben des Textbildschirms:

Black	= 0	Schwarz
Blue	= 1	Blau
Green	= 2	Grün
Cyan	= 3	Türkis
Red	= 4	Rot
Magenta	= 5	Purpur
Brown	= 6	Braun
LightGray	= 7	Hellgrau
DarkGray	= 8	Dunkelgrau
LightBlue	= 9	Hellblau
LightGreen	= 10	Hellgrün
LightCyan	= 11	Helltürkis
LightRed	= 12	Hellrot
LightMagenta	= 13	Hellpurpur
Yellow	= 14	Gelb
White	= 15	Weiß
Blink	= 128	Blinkend, falls der Treiber es erlaubt

Die Angaben von 0 bis 15 und 128 können nur für den Textvordergrund gewählt werden, für den Hintergrund sind nur die Konstanten mit den Werten 0 bis 7 möglich.
Die nächsten Konstanten dienen der Abfrage der Möglichkeit des Bildschirmtreibers:

cpUnderLine	= $0001	Bildschirmtreiber erlaubt unterstrichenen Text.
cpBlink	= $0002	Bildschirmtreiber erlaubt blinkenden Text.
cpColor	= $0004	Bildschirmtreiber ist farbfähig.
cpChangeFont	= $0008	Bildschirmtreiber unterstützt das Austauschen der Bildschirmschriftart.
cpChangeMode	= $0010	Bildschirmtreiber erlaubt das Ändern des Bildschirmmodus.
cpChangeCursor	= $0020	Bildschirmtreiber erlaubt das Ändern des Aussehens des Textcursors.

Die folgenden Konstanten definieren das Aussehen des Text-Eingabecursors:

crHidden	= 0	Cursor ausgeblendet.
crUnderLine	= 1	Cursor nur unten (unterstrichen).
crBlock	= 2	Blockcursor.
crHalfBlock	= 3	Halber Blockcursor.

Fehlerkonstanten:

errOk = 0	Kein Fehler
vioOK = 0	Kein Fehler aufgetreten
ErrorCode: LongInt = ErrOK	Der von der letzten Operation zurückgemeldete Fehlercode.

Der Fehlerhandler:

ErrorHandler: TErrorHandler = @DefaultErrorHandler	Die Variable *ErrorHandler* kann auf eine einige Fehlerbehandlungsroutine gesetzt werden. Der Zeiger wird in der Voreinstellung auf die Funktion *DefaultErrorHandler* gesetzt.
ErrorInfo : Pointer = nil	Zeiger auf die erweiterten Fehlerinformationen.

Allgemeine Fehlerkonstanten:

errVioBase	= 1000	Der Grundwert für Videofehler.
errVioInit	= errVioBase + 1	Bildschirmtreiber-Initialisierung ist fehlgeschlagen.
errVioNotSupported	= errVioBase + 2	Nicht unterstützte Bildschirmfunktion.
errVioNoSuchMode	= errVioBase + 3	Ungültiger Bildschirmmodus.

Die Unit arbeitet auch mit unterschiedlichen Schriften, dafür sind die Codepages einmal allgemein und einmal als VGA-Codepages definiert:

iso_codepages = [iso01, iso02, iso03, iso04, iso05, iso06, iso07, iso08, iso09, iso10, iso13, iso14, iso15]	*iso_codepages* ist eine Aufzählung mit allen Codepages basierend auf der ISO-Codierung.
vga_codepages = [cp437, cp850, cp852, cp866]	*vga_codepages* ist eine Aufzählung mit allen Codepages, bei denen mit normaler VGA-Karte ein VGA-Font ausgewählt werden kann. Zu beachten ist, daß dabei die Grafikzeichen an falschen Stellen in der Tabelle sind.

Einschränkungen des Zeichensatzes:

LowAscii = true	Auf einigen Systemen sind die unteren 32 Zeichen (0 bis 31) der DOS-Codepage für die ASCII-Steuerzeichen erforderlich und können von Programmen nicht angezeigt werden.

	Ist *LowAscii* auf *True* gesetzt, können die unteren 32 ASCII-Zeichen genutzt werden, ist der Wert auf *False* gesetzt, sollten sie vermieden werden. *LowAscii* kann je nach Bedarf als Konstante, Variable oder Property definiert werden, es darf aber unter keinen Umständen erwartet werden, daß *LowAscii* geschrieben werden kann oder die Adresse geholt werden kann.
NoExtendedFrame = false	Der VT100-Zeichensatz kennt nur Linienzeichen mit einer einfachen Linie. Ist dieser Wert auf *True* gesetzt, werden die Zeichen zum Zeichnen von Linien automatisch immer auf die einfachen Linien gesetzt und die doppelten Linienzeichen werden automatisch konvertiert. *NoExtendedFrame* kann je nach Bedarf als Konstante, Variable oder Property definiert werden, es darf aber unter keinen Umständen erwartet werden, daß *NoExtendedFrame* geschrieben werden oder die Adresse geholt werden kann.

Bildschirmdaten:

ScreenHeight: Word = 0	Die aktuelle Bildschirmhöhe, mit 0 vorbelegt.
ScreenWidth : Word = 0	Die aktuelle Bildschirmbreite, mit 0 vorbelegt.
FVMaxWidth = 132	Die größtmögliche Bildschirmpufferbreite in Zeichen.

Typdeklarationen

```
PVideoBuf = ^TVideoBuf;
```

Zeiger auf *TVideoBuf*.

```
PVideoCell = ^TVideoCell;
```

Zeiger auf *TVideoCell*.

```
PVideoMode = ^TVideoMode;
```

Zeiger auf den Record *TVideoMode*.

```
TEncoding = (cp437, cp850, cp852, cp866, koi8r, iso01, iso02,
             iso03, iso04, iso05, iso06, iso07, iso08, iso09,
             iso10, iso13, iso14, iso15);
```

Dieser Datentyp ist nur auf unixartigen Betriebssystemen verfügbar.

Wert	Beschreibung
cp437	Codepage 437 (Englisch, US-amerikanischer Standard)
cp850	Codepage 850 (Westeuropa, Eingabeaufforderungen bei deutscher Windows-Version)
cp852	Codepage 852 (Mitteleuropa)
cp866	Codepage 866 (Russisch)
iso01	ISO 8859-1 (Latin 1, Mittel-/Westeuropa ohne Euro-Symbol)
iso02	ISO 8859-2 (Latin 2, Mitteleuropa, beispielsweise Polen)
iso03	ISO 8859-3 (Latin 3, Südeuropa)
iso04	ISO 8859-4 (Latin 4, Baltikum)
iso05	ISO 8859-5 (Kyrillisch)

Tabelle 04.4: Aufzählungswerte für den Datentyp TEncoding (Teil1 von 2)

Wert	Beschreibung
iso06	ISO 8859-6 (Arabisch)
iso07	ISO 8859-7 (Griechisch)
iso08	ISO 8859-8 (Hebräisch)
iso09	ISO 8859-9 (Latin 5, Türkisch)
iso10	ISO 8859-10 (Latin 6, Nordisch)
iso13	ISO 8859-13 (Latin 7, Baltikum)
iso14	ISO 8859-14 (Latin 8, Westeuropa)
iso15	ISO 8859-15 (Latin 9, ISO 8859-1 mit Euro-Symbol, Mitteleuropa)
koi8r	Codepage KOI8-R

Tabelle 04.4: Aufzählungswerte für den Datentyp TEncoding (Teil 2 von 2)

```
TErrorHandler = function(Code: LongInt;
                         Info: Pointer): TErrorHandlerReturnValue;
```

Mit der Funktion *TErrorHandler* wird eine eigene Fehlerbehandlungsfunktion registriert, diese muß einen der Werte aus Tabelle 04.5 zurückgeben. *Code* sollte den Fehlercode für die Fehlerbedingung enthalten und der Parameter *Info* beliebige Daten, die spezifisch für den Fehlercode sind, der an die Funktion übergeben wird.

```
TErrorHandlerReturnValue = (errRetry, errAbort, errContinue);
```

TErrorHandlerReturnValue gibt Fehler bekannt und reagiert entsprechend auf Fehlerbedingungen. Die Bedeutung der Felder des Aufzählungstyps ist in Tabelle 04.5 erläutert.

Wert	Beschreibung
errAbort	Abbrechen und Fehlercode zurückgeben.
errContinue	Abbrechen ohne einen Fehlercode zu melden.
errRetry	Operation wiederholen.

Tabelle 04.5: Die Aufzählungswerte für den Datentyp TErrorHandlerReturnValue

```
TVideoBuf = array[0..32759] of TVideoCell;
```

Der Datentyp *TVideoBuf* stellt den Bildschirm dar.

```
TVideoCell = Word;
```

TVideoCell beschreibt ein Zeichen auf dem Bildschirm. Eines der beiden Byte steht für das Farbattribut, mit dem das Zeichen auf dem Bildschirm angezeigt wird, das andere enthält den ASCII-Code des Zeichens, das abgebildet wird. Die genaue Position der unterschiedlichen Byte im Record ist betriebssystemspezifisch. Auf den meisten Litte-Endian-Systemen (= Intel-Format) enthält das High-Byte das Farbattribut und das Low-Byte den ASCII-Code des Zeichens selbst.

```
TVideoDriver = record
  InitDriver       : procedure;
  DoneDriver       : procedure;
  UpdateScreen     : procedure(Force: Boolean);
  ClearScreen      : procedure;
  SetVideoMode     : function(const Mode: TVideoMode): Boolean;
  GetVideoModeCount: function: Word;
  GetVideoModeData : function(Index: Word; var Data: TVideoMode): Boolean;
  SetCursorPos     : procedure(NewCursorX: Word; NewCursorY: Word);
  GetCursorType    : function: Word;
  SetCursorType    : procedure(NewType: Word);
  GetCapabilities  : function: Word;
end;
```

Über *TVideoDriver* und mit dem Aufruf *SetVideoDriver* kann ein eigener Bildschirmtreiber installiert werden. Bei der Funktion und im allgemeinen Abschnitt zu eigenen Bildschirmtreibern ab Seite O76 befinden sich weitere Informationen zu diesem Record.

```
TVideoMode = record
  Col  : Word;
  Row  : Word;
  Color: Boolean;
end;
```

Der Record *TVideoMode* beschreibt einen Bildschirmmodus. Seine Felder sind selbsterklärend: *Col* und *Row* steht für die Zahl der Spalten und Zeilen auf dem Bildschirm, *Color* ist *True,* wenn der Modus Farben unterstützt und *False*, wenn nicht.

```
TVideoModeSelector = function(const VideoMode: TVideoMode;
                              Params: LongInt): Boolean;
```

TVideoModeSelector ist der Prototyp eines Callbacks für die Auswahl des Bildschirmmodus.

Variablen

Die Unit *Video* enthält einige wichtige Variablen:

CursorLines: Byte;	*CursorLines* ist eine Bitmaske, die festlegt, welche Cursorzeilen sichtbar sind oder ausgeblendet werden. Jedes Bit steht für eine Zeile des Cursors. Diese Variable ist nicht auf allen Plattformen vorhanden und sollte deshalb mit Bedacht angewandt werden.
CursorX: Word;	Die aktuelle horizontale Position auf dem Bildschirm, an die die Daten geschrieben werden.
CursorY: Word;	Die aktuelle vertikale Position auf dem Bildschirm, an die die Daten geschrieben werden.
OldVideoBuf: PVideoBuf;	Das Array *OldVideoBuf* enthält den Inhalt des Bildschirms nach seiner letzten Aktualisierung. Die Funktion *UpdateScreen* prüft dieses Array und entscheidet dann, welche Zeichen auf dem Bildschirm aktualisiert werden müssen und welche nicht. Es ist zu beachten, daß das Array *OldVideoBuf* unter Umständen von einigen Bildschirmtreibern übergangen wird und deshalb möglichst vom Anwendungsentwickler nicht genutzt werden sollte. Das Array befindet sich hauptsächlich im Interface-Bereich der Unit *Video*, damit es beim Schreiben von Treibern zur Verfügung steht.
ScreenColor: Boolean;	*ScreenColor* gibt an, ob der aktuelle Bildschirm Farben unterstützt.
VideoBuf: PVideoBuf;	Diese Variable stellt das Herz der Unit *Video* dar und das Array ist der physikalische Bildschirm. Das Schreiben in das Array und anschließende Aufrufen von *UpdateScreen* zeigt die aktuellen Zeichen auf dem Monitor an.
VideoBufSize: LongInt;	Die aktuelle Größe des Bildschirmpuffers, auf den *VideoBuf* zeigt.

Die beiden folgenden Variablen sind nur für internen Gebrauch vorgesehen:

```
external_codepage: TEncoding;
internal_codepage: TEncoding;
```

4.20.3 Prozeduren und Funktionen

ClearScreen

```
procedure ClearScreen;
```

ClearScreen löscht den aktuellen Bildschirm und ruft anschließend *UpdateScreen* auf. Beim Löschen des Bildschirminhalts wird in alle Zeichenzellen des Bildschirmpuffers ein Leerzeichen und in die Farbzellen die voreingestellte Farbe (Hellgrau auf Schwarz, Farbattribut $07) geschrieben.
Siehe auch: *InitVideo* und *UpdateScreen*.

```
program testvideo;
uses
  video, keyboard, vidutil;
var
  i: LongInt;
  k: TKeyEvent;
begin
  InitVideo;
  InitKeyboard;
  for I := 1 to 10 do
    TextOut(i, i, 'Beliebige Taste löscht den Bildschirm');
  UpdateScreen(False);
  K := GetKeyEvent;
  ClearScreen;
  TextOut(1, 1, 'Bildschirm ist gelöscht. Ende mit bel. Taste');
  UpdateScreen(True);
  K := GetKeyEvent;
  DoneKeyBoard;
  DoneVideo;
end.
```

Listing 04.38: videoex/ex3.pp

DefaultErrorHandler

```
function DefaultErrorHandler(AErrorCode: LongInt; AErrorInfo: Pointer
                             ): TErrorHandlerReturnValue;
```

DefaultErrorHandler ist die voreingestellte Fehlerbehandlungsroutine des Bildschirmtreibers. Hierbei werden die Fehlercode-Angaben in *AErrorCode* und *AErrorInfo* in die globalen Variablen *ErrorCode* und *ErrorInfo* geschrieben und *errContinue* zurückgegeben.

DoneVideo

```
procedure DoneVideo;
```

DoneVideo schaltet den Bildschirmtreiber ab, wenn er aktiv ist. Ist er bereits abgeschaltet oder noch nicht initialisiert, kehrt die Prozedur sofort zurück. Beim Abschalten des Bildschirmtreibers werden alle belegten Ressourcen freigegeben und – falls möglich – der Bildschirm wird in den Status versetzt, den er vor dem Aufruf von *InitVideo* hatte. Zusätzlich sind die beiden Speicherbereiche *VideoBuf* und *OldVideoBuf* nach dem *DoneVideo* nicht mehr gültig.
DoneVideo sollte nach einem Aufruf von *InitVideo* immer die Arbeit abschließen. Schlägt der Aufruf fehl oder beim Programmende vergessen, wird der Bildschirm nach dem Beenden des Programms in einem unbrauchbaren Zustand hinterlassen.
Ein Beispiel für die Prozedur ist bei den meisten anderen Funktionen zu finden.
Fehler: Normalerweise sollten keine Fehler auftreten. Der Treiber meldet Fehler über die Variable *ErrorCode*.

Siehe auch: *InitVideo*.

GETCAPABILITIES

```
function GetCapabilities: Word;
```

GetCapabilities gibt die Fähigkeiten des aktuellen Treibers zurück, wobei es sich eine mit *OR* verknüpfte Kombination der folgenden Konstanten handelt:

cpUnderLine	Der Bildschirmtreiber unterstützt das Attribut für Unterstreichen.
cpBlink	Der Bildschirmtreiber unterstützt das Blinkend-Attribut.
cpColor	Der Bildschirmtreiber unterstützt Farbausgaben.
cpChangeFont	Der Bildschirmtreiber erlaubt das Ändern der Bildschirmschriftart.
cpChangeMode	Der Bildschirmtreiber erlaubt Modusänderungen.
cpChangeCursor	Der Bildschirmtreiber erlaubt das Ändern des Cursors.

Siehe auch: *GetCursorType* und *GetVideoDriver*.

```
program Example4; (* Beispiel für die Funktion GetCapabilities *)
uses Video;
var
  W: Word;

procedure TestCap(Cap: Word; Msg: String);
begin
  Write(Msg, ': ');
  if W and Cap = Cap then WriteLn('Ja') else WriteLn('Nein');
end;

begin
  W := GetCapabilities;
  WriteLn('Der Bildschirmtreiber unterstützt folgende Funktionen:');
  TestCap(cpUnderLine,    'Unterstrichene Zeichen');
  TestCap(cpBlink,        'Blinkende Zeichen     ');
  TestCap(cpColor,        'Farbige Zeichen       ');
  TestCap(cpChangeFont,   'Fontänderungen        ');
  TestCap(cpChangeMode,   'Videomodus-Änderung   ');
  TestCap(cpChangeCursor, 'Ändern der Cursorform ');
end.
```

Listing 04.39: videoex/ex4.pp

GETCURSORTYPE

```
function GetCursorType: Word;
```

GetCursorType gibt den aktuellen Cursortyp zurück, was einer der folgenden Werte ist:

crHidden	Unsichtbarerer/ausgeblendeter Cursor.
crUnderLine	Cursor aus einzelner Linie.
crBlock	Blockcursor.
crHalfBlock	Halber Blockcursor.

Zu beachten ist, daß nicht alle Treiber alle Arten von Cursordarstellung unterstützen.
Siehe auch: *SetCursorType* und *GetCapabilities*.

```
program Example5; (* Beispiel für die Funktion GetCursorType *)
uses
  Video, Keyboard, VidUtil;
```

```
const
  Cursortypes: array[crHidden..crHalfBlock] of String =
               ('Versteckt', 'Unterstrichen', 'Block', 'Halber Block');
begin
  InitVideo;
  InitKeyboard;
  TextOut(1, 1, 'Cursortyp: ' + CursorTypes[GetCursorType]);
  TextOut(1, 2, 'Ende mit beliebiger Taste.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

Listing 04.40: videoex/ex5.pp

GetLockScreenCount

```
function GetLockScreenCount: Integer;
```

GetLockScreenCount gibt die aktuelle Sperrstufe an. Ist sie Null, aktualisiert *UpdateScreen* den Bildschirm.

Siehe auch: *LockScreenUpdate*, *UnlockScreenUpdate* und *UpdateScreen*.

```
program Example6;   (* Beispiel für die Funktion GetLockScreenCount *)
uses Video, Keyboard, Vidutil;
var
  I: LongInt;
  S: String;
begin
  InitVideo;
  InitKeyboard;
  TextOut(1, 1, 'Press key until new text appears.');
  UpdateScreen(False);
  Randomize;
  for I := 0 to Random(10) + 1 do LockScreenUpdate;
  I := 0;
  while GetLockScreenCount <> 0 do begin
    Inc(I);
    Str(I, S);
    UnlockScreenUpdate;
    GetKeyEvent;
    TextOut(1, 1, 'UnLockScreenUpdate had to be called ' + S + ' times');
    UpdateScreen(False);
  end;
  TextOut(1, 2, 'Press any key to end.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

Listing 04.41: videoex/ex6.pp

GetVideoDriver

```
procedure GetVideoDriver(var Driver: TVideoDriver);
```

GetVideoDriver ergibt den aktuellen Bildschirmtreiber-Record in *Driver*. Auf diese Weise kann einerseits der aktuelle Treiber geklont werden, andererseits können dann einzelne Bereiche mit *SetVideoDriver* überschrieben werden.

GetVideoMode

```
procedure GetVideoMode(var Mode: TVideoMode);
```

GetVideoMode gibt die Einstellungen des aktuellen Bildschirmmodus zurück. Die Felder *row* und *col* der Record-Variable *Mode* enthalten die Größe des aktuellen Bildschirms und *Mode.Color* den Wert *True*, wenn der Bildschirm farbfähig ist.
Siehe auch: *SetVideoMode* und *GetVideoModeData*.

```
program Example7;    (* Beispiel für die Funktion GetVideoMode *)
uses
  Video, Keyboard, Vidutil;
var
  M: TVideoMode;
  S: String;
begin
  InitVideo;
  InitKeyboard;
  GetVideoMode(M);
  if M.Color then
    TextOut(1, 1, 'Der aktuelle Modus unterstützt Farbe.')
  else
    TextOut(1, 1, 'Der aktuelle Modus kann keine Farben.');
  Str(M.Row, S);
  TextOut(1, 2, 'Zahl der Zeilen : ' + S);
  Str(M.Col, S);
  TextOut(1, 3, 'Zahl der Spalten: ' + S);
  TextOut(1, 4, 'Ende mit beliebiger Taste.');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

Listing 04.42: videoex/ex7.pp

GetVideoModeCount

```
function GetVideoModeCount : Word
```

GetVideoModeCount gibt die Zahl der Bildschirmmodi, die der aktuelle Treiber unterstützt, zurück. Erlaubt der Treiber das Umschalten der Bildschirmmodi nicht, retourniert die Funktion die Zahl 1.
In Verbindung mit *GetVideoModeData* können über diese Funktion die Daten der unterstützten Bildschirmmodi abgefragt werden.
Siehe auch: *GetVideoModeData* und *GetVideoMode*.

```
program Example8; (* Beispiel für die Funktion GetVideoModeCount *)
uses
  Video, Keyboard, VidUtil;

procedure DumpMode(M: TVideoMode; Index: Integer);
var
  S: String;
begin
  Str(Index:2, S);
  Inc(Index);
  TextOut(1, Index, 'Daten für den Modus ' + S + ': ');
  if M.Color then TextOut(19, Index, '      Farbe,')
             else TextOut(19, Index, 'Keine Farbe,');
  Str(M.Row:3, S);
```

```
  TextOut(28, Index, S + ' Zeilen');
  Str(M.Col:3, S);
  TextOut(36, Index, S + ' Spalten');
end;

var
  i, Count: Integer;
  m       : TVideoMode;
begin
  InitVideo;
  InitKeyboard;
  Count := GetVideoModeCount;
  for I := 1 to Count do begin
    GetVideoModeData(I - 1, M);
    DumpMode(M, I - 1);
  end;
  TextOut(1, Count + 1, 'Weiter mit beliebiger Taste');
  UpdateScreen(False);
  GetKeyEvent;
  DoneKeyboard;
  DoneVideo;
end.
```

Listing 04.43: videoex/ex8.pp

GETVIDEOMODEDATA

```
function GetVideoModeData(Index: Word; var Data: TVideoMode): Boolean;
```

GetVideoModeData liefert die Charakteristika des in *Index* ausgewählten Bildschirmmodus in *Data* zurück. *Index* ist nullbasiert und besitzt einen höchstmöglichen Wert von *GetVideoModeCount-1*. Unterstützt der aktuelle Treiber die Modus-Umschaltung nicht (*GetVideoModeCount=1*) und ist damit der *Index* automatisch 0, werden die Daten des aktuellen Modus zurückgegeben. Die Funktion meldet als Ergebnis *True*, falls die Modusdaten erfolgreich abgefragt werden konnten, sonst *False*.
Fehler: Falls für Index ein ungültiger Wert übergeben wird, ergibt die Funktion *False*.
Siehe auch: *GetVideoModeCount*, *SetVideoMode* und *GetVideoMode*.
Ein Beispiel für diese Funktion ist bei *GetVideoModeCount* gezeigt.

INITVIDEO

```
procedure InitVideo;
```

InitVideo initialisiert das Video-Subsystem. War es bereits eingeschaltet, macht die Prozedur nichts und kehrt sofort zurück. Nach dem Initialisieren des Treibers werden die beiden Zeiger *VideoBuf* und *OldVideoBuf* belegt, basierend auf den beiden Variablen *ScreenWidth* und *ScreenHeight*. Danach wird der Bildschirm gelöscht.
Fehler: Kann der Treiber nicht initialisiert werden, wird die Variable *ErrorCode* gesetzt.
Siehe auch: *DoneVideo*.
Ein Beispiel für diese Prozedur ist bei den meisten anderen Funktionen gezeigt.

LOCKSCREENUPDATE

```
procedure LockScreenUpdate;
```

LockScreenUpdate erhöht die Sperrstufe für die Bildschirmaktualisierung um den Wert 1. Solange dieser Zähler ungleich 0 ist, wird der Bildschirm nicht aktualisiert.
Mit dieser Funktion kann die Aktualisierung des Bildschirms optimiert werden. Insbesondere bei einer großen Zahl von Schreiboperationen auf den Bildschirm (möglicherweise von unbekannten Funktionen) führt der Aufruf von *LockScreenUpdate* vor

dem Zeichnen und *UnlockScreenUpdate* nach dem Zeichnen, gefolgt von einem *UpdateScreen* dazu, daß alle Schreibarbeiten auf den Bildschirm auf einen Schlag durchgeführt werden.
Siehe auch: *UpdateScreen*, *UnlockScreenUpdate* und *GetLockScreenCount*.
Ein Beispiel ist bei der Funktion *GetLockScreenCount* gezeigt.

SetCursorPos

```
procedure SetCursorPos(NewCursorX: Word; NewCursorY: Word);
```

SetCursorPos stellt den Cursor an die angegebene Position mit der Zeilenangabe *NewCursorX* und der Spalte *NewCursorY*. Der Ursprung der Bildschirmkoordinaten befindet sich mit dem Koordinatenpaar 0,0 an der linken oberen Ecke. Die aktuelle Position wird in den Variablen *CursorX* und *CursorY* gespeichert.
Siehe auch: *SetCursorType*.

```
program example2;
uses
  Video, Keyboard;
var
  P, PP, D: Integer;
  K       : TKeyEvent;

procedure PutSquare (P: Integer; C: Char);
begin
  VideoBuf^[P] := Ord(C) + ($07 shl 8);
  VideoBuf^[P + ScreenWidth] := Ord(c) + ($07 shl 8);
  VideoBuf^[P + 1] := Ord(c) + ($07 shl 8);
  VideoBuf^[P + ScreenWidth + 1] := Ord(c) + ($07 shl 8);
end;

begin
  InitVideo; InitKeyBoard;
  P  := 0;
  PP := -1;
  repeat
    if PP <> -1 then PutSquare(PP, ' ');
    PutSquare(P, '#');
    SetCursorPos(P mod ScreenWidth, P div ScreenWidth);
    UpdateScreen(False);
    PP := P;
    repeat
      D := 0;
      K := TranslateKeyEvent(GetKeyEvent);
      case GetKeyEventCode(K) of
        kbdLeft:  if P mod ScreenWidth) <> 0 then D := -1;
        kbdUp:    if P >= ScreenWidth then D := -ScreenWidth;
        kbdRight: if (P + 2) mod ScreenWidth <> 0 then D := 1;
        kbdDown:  if P < (VideoBufSize div 2) - (ScreenWidth * 2) then
          D := ScreenWidth;
      end;
    until (D <> 0) or (GetKeyEventChar(K) = 'q');
    P := P + D;
  until GetKeyEventChar(K) = 'q';
  DoneKeyBoard;
  DoneVideo;
end.
```

Listing 04.44: videoex/ex2.pp

SetCursorType

```
procedure SetCursorType(NewType: Word)
```

SetCursorType setzt das Aussehen des Cursors auf den Typ, der in *NewType* angegeben ist. Zur Auswahl stehen folgende Konstanten:

crHidden	Cursor ausblenden.
crUnderLine	Cursor als Unterstreichung.
crBlock	Blockcursor.
crHalfBlock	Cursor als halber Block.

Siehe auch: *SetCursorPos*.

SetVideoDriver

```
function SetVideoDriver(const Driver: TVideoDriver): Boolean;
```

SetVideoDriver wählt *Driver* als aktuellen Bildschirmtreiber aus. Ist der aktuelle Bildschirmtreiber bereits initialisiert, wurde also *InitVideo* bereits aufgerufen, macht die Funktion nichts, sondern kehrt sofort zurück und gibt den Wert *False* als Funktionsergebnis aus. Um einen neuen Treiber zu installieren, muß also vorher *DoneVideo* aufgerufen werden.
Ein Beispiel ist im Abschnitt zum Schreiben eigener Bildschirmtreiber ab Seite O76 gezeigt.
Fehler: Ist der aktuelle Treiber bereits installiert, gibt die Funktion *False* zurück.

SetVideoMode

```
function SetVideoMode(const Mode: TVideoMode): Boolean;
```

SetVideoMode setzt den in *Mode* angegebenen Bildschirmmodus. War der Aufruf erfolgreich, besitzt der Bildschirm *Mode.Col* Zeilen und *Mode.Row* Spalten und wird in Farbe angezeigt, wenn *Mode.Color* den Wert *True* hat.
Die Funktion meldet *True*, wenn der Modus erfolgreich eingeschaltet werden konnte, *False*, wenn nicht.
Bemerkung: Der Bildschirmmodus muß nicht unbedingt gesetzt werden, zum Beispiel können eine Konsole unter Linux oder eine Telnet-Sitzung den Modus nicht immer setzen. Es muß also überprüft werden, ob der Fehlerwert, den die Funktion zurückgibt, einen Fehler signalisiert.
Der Modus kann bereits festgelegt werden, wenn der Bildschirmtreiber noch gar nicht initialisiert wurde, das heißt, wenn *InitVideo* noch gar nicht aufgerufen wurde. In diesem Fall wird der gewählte Modus nach dem Initialisieren des Treibers mit *InitVideo* eingeschaltet. Der Aufruf von *SetVideoMode* vor dem Aufruf von *InitVideo* löscht den voreingestellten Bildschirmmodus.
Um zu erfahren, welche Modi überhaupt gültig sind, müssen die Funktionen *GetVideoModeCount* und *GetVideoModeData* aufgerufen werden, der aktuelle Bildschirmmodus wird mit *GetVideoMode* ermittelt.
Fehler: Kann der gewünschte Bildschirmmodus nicht gesetzt werden, wird in *ErrorCode* der Wert *errVioNoSuchMode* gesetzt.
Siehe auch: *GetVideoModeCount*, *GetVideoModeData*, *GetVideoMode*

UnlockScreenUpdate

```
procedure UnlockScreenUpdate
```

UnlockScreenUpdate verringert den Sperrzähler um Eins, wenn er größer Null ist. Erreicht der Zähler den Wert 0, aktualisiert ein Aufruf von *UpdateScreen* den Bildschirm-

inhalt wieder. Solange der Zähler größer Null ist, wird der Bildschirm nicht aktualisiert.
Mit Hilfe dieses Mechanismus wird die Geschwindigkeit bei zahlreichen Bildschirmausgaben deutlich erhöhlt. Dabei muß aber sichergestellt sein, daß die Zahl der Aufrufe von *LockScreenUpdate* genau zur Zahl der Aufrufe von *UnlockScreenUpdate* paßt.
Siehe auch: *LockScreenUpdate*, *GetLockScreenCount* und *UpdateScreen*.
Ein Beispiel ist bei *GetLockScreenCount* gezeigt.

UpdateScreen

```
procedure UpdateScreen(Force: Boolean);
```

UpdateScreen synchronisiert den aktuellen Bildschirm mit dem Inhalt des internen Puffers *VideoBuf*. Der Parameter *Force* legt fest, ob der komplette Bildschirm neu gezeichnet werden soll (bei *Force=True*), oder ob nur die Teile des Puffers auf den Monitor geschrieben werden sollen, die sich seit der letzten Aktualisierung geändert haben.
Die Unit *Video* bewahrt eine interne Kopie des letzten Bildschirms im Array *OldVideoBuf*. Der aktuelle Inhalt von *VideoBuf* dagegen wird bei der Prüfung, welche Teile neu geschrieben werden müssen, abgeglichen. Auf langsamen Terminals wie beispielsweise einer Linux-Telnet-Sitzung beschleunigt dieser Mechanismus die Bildschirmausgabe merklich.
Bei Plattformen, bei denen die Sichtbarkeit des Mauscursors bei Bildschirmaktualisierungen nicht garantiert ist, muß der Mauszeiger nach dem Update aktualisiert werden. Normalerweise wird dazu vor dem Schreiben die Prozedur *HideMouse* und nach dem Update die Routine *ShowMouse*, beide aus der Unit *Mouse*, aufgerufen.
Ein Beispiel für diese Prozedur ist bei den meisten anderen Funktionen abgebildet.
Siehe auch: *ClearScreen*.

4.21 Unit Mouse

Die Unit *Mouse* bietet eine plattformunabhängige Schnittstelle zur Maussteuerung, die auf allen von Free Pascal 2 unterstützten Plattformen verfügbar ist. Sie kann bei Bedarf durch selbstgeschriebene Treiber erweitert werden, beispielsweise um durch das Protokollieren von Mausereignissen eine Aufzeichnungs- und Wiederabspielfunktion zur Verfügung zu stellen.
Die Unit ist nur für Textmodus- und nicht für grafische Programme ausgelegt und basiert prinzipiell auf dem Zusammenspiel mit den beiden Units *Keyboard* und *Video*.
Wichtig: Die Unit Mouse von Free Pascal 2 ist nicht zur gleichnamigen Unit von Free Pascal 1 kompatibel!

4.21.1 Konstanten, Typdeklarationen und Variablen

Konstanten

Fehlerkonstanten:

Konstante	Wert	Beschreibung
errMouseBase	1030	Basis für die Fehlercodes der Maussteuerung.
errMouseInitError	errMouseBase + 0	Initialisierungsfehlercode der Maus.
errMouseNotImplemented	errMouseBase + 1	Maustreiber nicht implementiert.

Allgemeine Konstanten für die Maussteuerung:

Konstante	Wert	Beschreibung
MouseActionDown	$0001	Signal, wenn die Maustaste gedrückt wird.
MouseActionMove	$0004	Mausbewegungsevent.
MouseActionUp	$0002	Signal, wenn die Maustaste losgelassen wird.
MouseEventBufSize	16	Die Unit Mouse besitzt einen Mechanismus für das Puffern der Mausereignisse, diese Konstante legt die Größe dieses Puffers fest.

Button-Events:

Konstante	Wert	Beschreibung
MouseLeftButton	$01	Linke Maustaste.
MouseMiddleButton	$04	Mittlere Maustaste.
MouseRightButton	$02	Rechte Maustaste.

Datentypen

```
PMouseEvent = ^TMouseEvent;
```

Zeiger auf den Record *TMouseEvent*.

```
TMouseDriver = record
  UseDefaultQueue: Boolean;
  InitDriver     : procedure;
  DoneDriver     : procedure;
  DetectMouse    : function: Byte;
  ShowMouse      : procedure;
  HideMouse      : procedure;
  GetMouseX      : function: Word;
```

```
  GetMouseY      : function: Word;
  GetMouseButtons: function: Word;
  SetMouseXY     : procedure(x: Word; y: Word);
  GetMouseEvent  : procedure(var MouseEvent: TMouseEvent);
  PollMouseEvent : function(var MouseEvent: TMouseEvent): Boolean;
  PutMouseEvent  : procedure(const MouseEvent: TMouseEvent);
end;
```

Mit dem Record *TMouseDriver* wird ein Maustreiber in der Funktion *SetMouseDriver* definiert. Seine Felder müssen vor dem Aufruf der Funktion *SetMouseDriver* gefüllt sein.

```
TMouseEvent = packed record
  buttons: Word;
  x      : Word;
  y      : Word;
  Action : Word;
end;
```

Das Ereignis *TMouseEvent* ist der zentrale Datentyp der Unit *Mouse*, es beschreibt alle Ereignisse. Das Feld *Buttons* beschreibt, welche Buttons beim Auftreten des Ereignisses gedrückt waren, die Felder *x* und *y* enthalten die Position, an der das Ereignis eintrat. *Action* gibt an, was der Fall war, als das Ereignis eintrat. Die Felder *Buttons* und *Actions* können auf die Konstanten im Interface der Unit untersucht werden.

Variablen

MouseButtons: Byte	Diese Variable speichert die Position des letzten bekannten Mausbuttonstatus. Sie sollte nicht verwendet werden.
MouseIntFlag: Byte	Diese Variable speichert die Position des letzten internen Mausbuttonstatus. Sie sollte nicht verwendet werden.
MouseWhereX: Word	Diese Variable speichert die Position der letzten bekannten X-Cursorposition. Sie sollte nicht verwendet werden.
MouseWhereY: Word	Diese Variable speichert die Position des letzten bekannten Y-Cursorposition. Sie sollte nicht verwendet werden.

4.21.2 Prozeduren und Funktionen

DetectMouse

```
function DetectMouse: Byte;
```

DetectMouse stellt fest, ob am System eine Maus angeschlossen ist. Wird keine Maus gefunden, wird der Wert 0 zurückgeliefert. Ist die Suche erfolgreich, meldet die Funktion die Zahl der Maustasten.
Diese Funktion sollte aufgerufen werden, nachdem der Maustreiber initialisiert wurde.
Siehe auch: *InitMouse* und *DoneMouse*.

```
program Example1;    (* Beispiel für die Funktion DetectMouse *)
uses
  Mouse;
var
  Buttons: Byte;
begin
  InitMouse;
  Buttons := DetectMouse;
  if Buttons = 0 then WriteLn('Keine Maus angeschlossen.')
                 else WriteLn(Buttons, '-Tasten-Maus gefunden.');
```

```
  DoneMouse;
end.
```

Listing 04.45: mouseex/ex1.pp

DoneMouse

```
procedure DoneMouse
```

DoneMouse fährt den Maustreiber herunter, wobei der vom Treiber belegte Speicher freigegeben oder mögliche Maushooks aus dem Speicher entfernt werden. Die Funktionen der Unit *Mouse* funktionieren nach diesem Aufruf nicht mehr; wird die Prozedur ein zweites Mal aufgerufen, kehrt sie sofort zurück. Bevor *DoneMouse* erneut aufgerufen werden kann, muß *InitMouse* ein weiteres Mal aufgerufen werden.
Ein Beispiel für diese Prozedur ist bei den meisten anderen Mausfunktionen gezeigt.
Siehe auch: *DetectMouse* und *InitMouse*.

GetMouseButtons

```
function GetMouseButtons : Word;
```

GetMouseButtons meldet den aktuellen Tastenstatus der Maus, das heißt, sie gibt eine mit *OR* verbundene Kombination der folgenden Konstanten zurück:

MouseLeftButton	Die linke Maustaste ist gedrückt.
MouseRightButton	Die rechte Maustaste ist gedrückt
MouseMiddleButton	Die mittlere Maustaste ist gedrückt

Siehe auch: *GetMouseEvent*, *GetMouseX*, *GetMouseY*.

```
program Example2;   (* Beispiel für die Funktion GetMouseButtons *)
uses
  mouse;
begin
  InitMouse;
  WriteLn('Rechte Maustaste beendet das Programm');
  while GetMouseButtons <> MouseRightButton do { Dummy };
  DoneMouse;
end.
```

Listing 04.46: mouseex/ex2.pp

GetMouseDriver

```
procedure GetMouseDriver(var Driver: TMouseDriver);
```

GetMouseDriver ergibt den aktuell gesetzten Maustreiber. Auf diese Weise kann der aktuelle Treiber ermittelt werden, damit verschiedene Callback-Routinen überschrieben werden können.
Siehe auch: *SetMouseDriver*.

GetMouseEvent

```
procedure GetMouseEvent(var MouseEvent: TMouseEvent);
```

GetMouseEvent liefert das nächsten Mausereignis (eine Bewegung, einen Tastendruck oder eine Tastenfreigabe) oder wartet auf das Ereignis, wenn sich derzeit keines in der Warteschlange befindet.
Einige Maustreiber stellen eine Maus-Event-Warteschlange zur Verfügung, in der man mehrere Events gleichzeitig zwischenspeichern kann, andere machen das nicht. In solchen Fällen ist eine Event-Warteschlange für ein einziges Ereignis in *PollMouseEvent* verfügbar.
Siehe auch: *GetMouseButtons*, *GetMouseX* und *GetMouseY*.

GetMouseX

```
function GetMouseX : Word;
```

GetMouseX gibt die aktuelle horizonatale Position der Maus auf dem Textbildschirm an, dabei wird mit dem Wert 0 an der linken Seite des Bildschirms begonnen.
Siehe auch: *GetMouseButtons*, *GetMouseEvent* und *GetMouseY*.

```
program Example4;   (* Beispiel für die Funktionen GetMouseX/GetMouseY *)
uses
  Mouse;
var
  X, Y: Word;
begin
  InitMouse;
  WriteLn('Maus in das Quadrat 10,10 bewegen um zu beenden');
  repeat
    X := GetMouseX;
    Y := GetMouseY;
    WriteLn('X,Y= (',X, ',' ,Y, ')');
  until (X = 9) and (Y = 9);
  DoneMouse;
end.
```

Listing 04.47: mouseex/ex4.pp

GetMouseY

```
function GetMouseY: Word;
```

GetMouseY gibt die vertikale Position der Maus auf dem Bildschirm zurück. *Y* wird in Zeichen angegeben, wobei am oberen Bildschirmrand mit dem Wert 0 begonnen wird.
Ein Beispiel für diese Funktion ist bei *GetMouseX* gezeigt.
Siehe auch: *GetMouseButtons*, *GetMouseEvent* und *GetMouseX*.

HideMouse

```
procedure HideMouse
```

HideMouse blendet den Mauszeiger aus. Diese Funktion funktioniert nicht unbedingt auf allen Systemen. Ob sie Auswirkungen zeitigt, hängt vom Treiber ab.
Siehe auch: *ShowMouse*.

```
program Example5;    (* Beispiel für die Funktion HideMouse *)
uses
  Mouse;
var
  Event  : TMouseEvent;
  Visible: Boolean;
begin
  InitMouse;
  ShowMouse;
  Visible := true;
  WriteLn('Linke Maustaste zeigt/versteckt, rechte Taste beendet');
  repeat
    GetMouseEvent(Event);
    with Event do
      if (Buttons = MouseLeftbutton) and
         (Action = MouseActionDown) then begin
        if Visible then HideMouse else ShowMouse;
        Visible := not Visible;
      end;
  until (Event.Buttons = MouseRightButton) and
```

```
        (Event.Action = MouseActionDown);
  DoneMouse;
end.
```

Listing 04.48: mouseex/ex5.pp

InitMouse

```
procedure InitMouse;
```

InitMouse initialisiert den Maustreiber. Die Prozedur belegt alle benötigten Datenstrukturen, damit die Maus funktioniert. Nach dem *InitMouse* können die anderen Mausfunktionen aufgerufen werden.
Einem Aufruf von *InitMouse* muß beim Programmende immer ein Aufruf von *DoneMouse* folgen. Wird er ausgelassen, ist die Maus nicht mehr brauchbar oder es kommt sogar zu Speicherlöchern.
Ein Beispiel für diese Prozedur ist bei den meisten anderen Mausfunktionen gezeigt.
Siehe auch: *DoneMouse* und *DetectMouse*.

PollMouseEvent

```
function PollMouseEvent(var MouseEvent: TMouseEvent): Boolean;
```

PollMouseEvent prüft, ob ein Mausereignis verfügbar ist und gibt es, wenn das der Fall ist, in *MouseEvent* zurück. Das Funktionsergebnis ist in diesem Fall *True*. Liegt kein Mausereignis an, ist das Funktionsergebnis *False* und der Inhalt von *MouseEvent* undefiniert.
Zu beachten ist, daß das Ereignis nach dem Aufruf von *PollMouseEvent* nicht aus der Warteschlange entfernt ist, es muß erst mit *GetMouseEvent* gelöscht werden.
Siehe auch: *GetMouseEvent* und *PutMouseEvent*.

PutMouseEvent

```
procedure PutMouseEvent(const MouseEvent: TMouseEvent);
```

PutMouseEvent fügt dem Eingabepuffer das Ereignis *MouseEvent* hinzu. Der nächste Aufruf von *GetMouseEvent* oder *PollMouseEvent* gibt diesen *MouseEvent* dann zurück.
Es ist zu beachten, daß, abhängig von ihrem Aufbau, die Mausereigniswarteschlange eventuell nur einen Wert zwischenspeichern kann.
Siehe auch: *GetMouseEvent* und *PollMouseEvent*.

SetMouseDriver

```
procedure SetMouseDriver(const Driver: TMouseDriver);
```

SetMouseDriver setzt den Maustreiber auf den Wert *Driver*. Diese Routine muß vor dem Aufruf von *InitMouse* und nach dem Aufruf von *DoneMouse* aufgerufen werden. Wird die Prozedur nach dem Initialisieren des Maustreibers aufgerufen, ist sie wirkungslos.
Siehe auch: *InitMouse*, *DoneMouse* und *GetMouseDriver*.

SetMouseXY

```
procedure SetMouseXY(x: Word; y: Word);
```

SetMouseXY plaziert den Mauszeiger auf dem Koordinatenpaar *X,Y*. *X* und *Y* sind nullbasierte Zeichenkoordinaten, *0,0* befindet sich an der oberen linken Ecke des Bildschirms. Die Position wird in Zeichenzellen und nicht in Bildpunkten gemessen.
Siehe auch: *GetMouseX* und *GetMouseY*.

```
program Example7;     (* Beispiel für die Funktion SetMouseXY *)
uses
  Mouse;
begin
  InitMouse;
  Writeln('Rechte Maustaste beendet das Programm');
```

```
  SetMouseXY(40, 12);
  repeat
    Writeln(GetMouseX, ',', GetMouseY);
    if GetMouseX > 70 then SetMouseXY(10, GetMouseY);
    if GetMouseY > 20 then SetMouseXY(GetMouseX, 5);
  until GetMouseButtons = MouseRightButton;
  DoneMouse;
end.
```

Listing 04.49: mouseex/ex7.pp

ShowMouse

```
procedure ShowMouse;
```

ShowMouse zeigt einen bisher ausgeblendeten Mauszeiger. Die Möglichkeit, ob das Aus- und Einblenden möglich ist, hängt von den Fähigkeiten des Treibers ab.
Ein Beispiel für diese Prozedur ist bei *HideMouse* gezeigt.
Siehe auch: *HideMouse*.

4.22 Unit Keyboard

Die Unit *Keyboard* stellt eine systemunabhängige Zugriffsschicht für die Tastatur zur Verfügung. Mit ihr können der Tastaturstatus abgefragt und auch diverse Ereignisse gewartet werden. Die Funktion *GetKeyEvent* wartet auf ein Tastaturereignis und liefert ein treiberabhängiges Tastenereignis. Dieser Event kann dann mit der Funktion *TranslateKeyEvent* in ein interpretierbares Ereignis umgewandelt werden. Das Ergebnis dieses Aufrufs kann dann in den anderen Untersuchungsfunktionen für Tastaturereignisse verarbeitet werden.

Ein eigener Tastaturtreiber wird mit der Funktion *SetKeyboardDriver* installiert, den aktuellen ermittelt *GetKeyboardDriver*.

Im letzten allgemeinen Abschnitt dieses Kapitels wird gezeigt, wie ein eigener Tastaturtreiber entwickelt wird.

Ein Teil der Funktionen in dieser Unit ist derzeit nicht dokumentiert, was jeweils entsprechend angegeben ist.

4.22.1 Spezielle Hinweise zu Unix

Unter Unix laufen Programme in einem »Terminal« und die Programme, die auf den Bildschirm schreiben und von der Tastatur lesen, kommunizieren mit dem Terminal. Die Tastaturbehandlung unter Unix ist größtenteils abwärtskompatibel zum DEC vt100 und vt220 von vor vielen Jahren. Diese Geräte hatten Tastaturen, die sich von denen heutiger PCs drastisch unterschieden und an dieser Stelle fangen die Probleme an. Verschlimmert wird die Situation dadurch, daß das Design des Protokolls der beiden Terminals außerdem nicht sehr gut war.

Unter diesem Hintergrund versucht die Unit *Keyboard*, auf Unix-Systemen die Tastaturfunktionalität bestmöglich zur Verfügung zu stellen. Eine Implementation mit allen Möglichkeiten anderer Betriebssysteme ist aufgrund des unterliegenden Designs nicht möglich.

Eine Ausnahme hiervon stellt der Linux-Kernel bereit. Die Terminalemulation des Linux-Kernels ist aus Sicht einer PC-Tastatur zwar ebenfalls hoffnungslos primitiv, aber im Gegensatz zu den anderen Terminal-Emulatoren voll konfigurierbar. Auf der Linux-Konsole versucht die Unit Keyboard von Free Pascal, den vollen Funktionsumfang zu implementieren.

Wird die Unit Keyboard in Programme eingebunden, besitzen diese einen vollen Funktionsumfang auf der Linux-Konsole. Es muß sich aber um eine nackte Konsole handeln, ein SSH in eine andere Maschine zerstört die volle Funktionalität. Andernfalls liegt ein eingeschränkter Funktionsumfang vor.

Einige Bemerkungen zum vollen Funktionsumgang bei Linux:

- Wird die Tastatur umprogrammiert und aus irgendeinem Grund nicht in ihren Originalzustand zurückversetzt, muß die Tastaturbelegung mit dem Kommandozeilenbefehl *reset* neu geladen werden, um sie zu reinitialisieren.
- [Alt] in Kombination mit Funktionstasten erzeugt Steuercodes für diese Tasten. Um zwischen virtuellen Konsolen umzuschalten, muß mit der Tastenkombination [Strg]+[Alt]+[Fxx] gearbeitet werden.
- Im Gegensatz zu anderer Unix-Software, erzeugt [Esc] eine Tastaturcode für [Esc] ohne Wartezeit.

Die eingeschränkte Funktionalität hat die systembedingten Probleme/Schwächen:

- Die [Esc]-Taste muß, damit sie wirksam ist, zweimal gedrückt werden.
- Wird auf der Linux-Konsole mit einem Programm gearbeitet, mit dem auf eine andere Maschine eingeloggt ist:

  - [Shift]+[F1] und [Shift]+[F12] erzeugen die Tastaturcodes für [F11] und [F12].
  - [Shift]+Pfeiltasten, [Shift]+[Einfg], [Shift]+[Entf], [Shift]+[Pos1] und [Shift]+[Ende] funktionieren nicht. Dies trifft auch für die Kombinationen mit [Alt] und [Strg] zu.
  - [Alt] in Kombination mit den Funktionstasten schaltet zwischen den Konsolen um liefert nicht die richtigen Tastatursequenzen.
  - [Strg] in Verbindung mit den Funktionstasten liefert die Steuercodes für die Funktionstasten ohne [Strg].
- In Xterm:
  - [Shift]+[Einfg] fügt die Daten des X-Clipboards ein, es wird kein Tastaturcode erzeugt.
- Auf der KDE-Konsole:
  - [Shift]+[Einfg] fügt die Daten aus dem X-Clipboard ein, es wird kein Tastaturcode generiert.
  - [Shift] in Verbindung mit den Pfeiltasten funktioniert ebensowenig wie die Pfeiltasten in Verbindung mit der [Strg]-Taste.

Ist eine Nicht-Standardtastatur angeschlossen, kann es sein, daß auch einige andere Tasten nicht funktionieren. Befindet man sich im Modus mit der eingeschränkten Funktionalität, kann aber mit dem üblichen Workaround um die Funktionstasten und deren ESC-Präfix herumgearbeitet werden:

- [Esc]+[1] = F1, [Esc]+[2] = F2 und so weiter.
- [Esc] vor der Eingabe einer anderen Taste ist gleichwertig mit der [Alt]+Taste.

In solchen Fällen und wenn das Terminal eine Ausgabe der ESC-Sequenzen für diese Tasten zur Verfügung stellt, bitten die Free-Pascal-Entwickler um einen Bugreport, damit das Terminal hinzugefügt werden kann.

4.22.2 Tastaturtreiber schreiben

Beim Schreiben eines Tastaturtreibers müssen für die meisten der Funktionen der Unit *Keyboards* Hooks erzeugt werden. Der Record *TKeyBoardDriver* enthält ein Feld für jeden der möglichen Hooks:

```
TKeyboardDriver = record
  InitDriver              : procedure;
  DoneDriver              : procedure;
  GetKeyEvent             : function : TKeyEvent;
  PollKeyEvent            : function : TKeyEvent;
  GetShiftState           : function : Byte;
  TranslateKeyEvent       : function(KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: function(KeyEvent: TKeyEvent): TKeyEvent;
end;
```

Die Bedeutung der Felder erläutert Tabelle O4.6.

Hook	Bedeutung
InitDriver	Wird aufgerufen, um den Treiber zu initialisieren und einzuschalten. Dieser Aufruf darf nur einmal erfolgen und muß dafür sorgen, daß alles, was der Treiber benötigt, initialisiert wird.
DoneDriver	Wird aufgerufen, um den Treiber abzuschalten und aufzuräumen. Dieser Aufruf muß garantiert nach einem *InitDriver* erfolgen und sollte alles, was von *InitDriver* initialisiert wurde, wieder zurücksetzen.
GetKeyEvent	Wird von *GetKeyEvent* aufgerufen. Muß auf das nächste Tastaturereignis warten und es zurückgeben. Die Funktion darf die Tasten nicht speichern.

Tabelle O4.6: Die Hooks im Record TKeyBoardDriver (Teil 1 von 2)

Hook	Bedeutung
PollKeyEvent	Wird von *PollKeyEvent* aufgerufen und muß die nächste Taste, wenn ein Event anliegt, zurückgeben. Darf die Tasten nicht speichern.
GetShiftState	Wird von *PollShiftStateEvent* aufgerufen und muß den aktuellen Status der Strg-, Alt- und Shift-Taste (Shiftstate) zurückgeben.
TranslateKeyEvent	Übersetzt einen rohen Tastaturevent in ein richtiges Tastaturereignis, das heißt, muß den Shiftstate eintragen und die Funktionstasten-Scancodes in Funktionstasten-Steuercodes umwandeln. Ist *TranslateKeyEvent* nicht eingetragen, wird eine voreingestellte Übersetzungsfunktion aufgerufen, die die die bekannten Scancodes aus Tabelle O4.7 in richtige Tastaturevents umsetzt.
TranslateKeyEvent-UniCode	Übersetzt ein Tastaturereignis in ein Unicode-Gegenstück.

Tabelle O4.6: Die Hooks im Record TKeyBoardDriver (Teil 2 von 2)

Genaugenommen müssen nur die Hooks *GetKeyEvent* und *PollKeyEvent* implementiert werden, damit der Treiber richtig funktioniert.
Folgende Beispiel-Unit *logkeys* zeigt, wie ein Tastaturtreiber installiert wird, wobei der installierte Treiber herangezogen wird und Hooks in die Funktion *GetKeyEvent* eingehängt werden, um die Tastaturevents zu registrieren und in einer Datei mitzuprotokollieren.

```
unit logkeys;

interface
procedure StartKeyLogging;
procedure StopKeyLogging;
function  IsKeyLogging: Boolean;
procedure SetKeyLogFileName(FileName: String);

implementation
uses SysUtils, Keyboard;
var
  NewKeyBoardDriver,
  OldKeyBoardDriver: TKeyboardDriver;
  Active,Logging   : Boolean;
  LogFileName      : String;
  KeyLog           : Text;

function TimeStamp: String;
begin
  TimeStamp := FormatDateTime('hh:nn:ss', Time());
end;

procedure StartKeyLogging;
begin
  Logging := True;
  Writeln(KeyLog, 'Start logging keystrokes at: ', TimeStamp);
end;

procedure StopKeyLogging;
begin
  Writeln(KeyLog, 'Stop logging keystrokes at: ', TimeStamp);
  Logging := False;
end;
```

```
function IsKeyLogging: Boolean;
begin
  IsKeyLogging := Logging;
end;

function LogGetKeyEvent: TKeyEvent;
var
  K: TKeyEvent;
begin
  K := OldkeyboardDriver.GetKeyEvent();
  if Logging then begin
    Write(KeyLog,TimeStamp, ': Key event: ');
    Writeln(KeyLog, KeyEventToString(TranslateKeyEvent(K)));
  end;
  LogGetKeyEvent := K;
end;

procedure LogInitKeyBoard;
begin
  OldKeyBoardDriver.InitDriver();
  Assign(KeyLog,logFileName); Rewrite(KeyLog);
  Active := True;
  StartKeyLogging;
end;

procedure LogDoneKeyBoard;
begin
  StopKeyLogging;
  Close(KeyLog);
  Active := False;
  OldKeyBoardDriver.DoneDriver();
end;

procedure SetKeyLogFileName(FileName: String);
begin
  if not Active then LogFileName := FileName;
end;

initialization
  GetKeyBoardDriver(OldKeyBoardDriver);
  NewKeyBoardDriver := OldKeyBoardDriver;
  NewKeyBoardDriver.GetKeyEvent := @LogGetKeyEvent;
  NewKeyBoardDriver.InitDriver  := @LogInitKeyboard;
  NewKeyBoardDriver.DoneDriver  := @LogDoneKeyboard;
  LogFileName := 'keyboard.log';
  Logging := False;
  SetKeyboardDriver(NewKeyBoardDriver);
end.
```

Dieser Treiber kann über einen anderen übergestülpt werden, solange er in der Uses-Klausel *nach* der richtigen Treiber-Unit eingebunden wird und wenn der echte Treiber in seinem Initialisierungsbereich den Treiber-Record setzt.

Durch Erweiterung dieser Beispiel-Unit kann ein Treiber geschrieben werden, der Tastendrücke aufnehmen und später auch wieder abspielen kann; auf diese Weise können Tastaturmakros aufgenommen und abgespielt werden. Dieser Treiber wiederum kann auf jeden anderen aufgesetzt werden.

Tasten-Scancodes

Eine große Zahl besonderer Tasten wird mit ihren DOS-Scancodes im zweiten Byte des Datentyps *TKeyEvent* belegt. Eine vollständige Liste aller Scancodes (als hexadezimale Werte) steht in Tabelle O4.7, wobei es sich dabei um die Liste der Tasten handelt, wie sie im voreingestellten Tastenübersetzungsmechanismus angegeben sind. Beim Schreiben eines Tastaturtreibers müssen diese Konstanten entweder von den verschiedenen Tasten-Event-Funktionen zurückgemeldet werden oder es muß der Hook *TranslateKeyEvent* vom Treiber implementiert werden.

Code	Taste	Code	Taste	Code	Taste
00	NoKey (Keine Taste)	3D	F3	70	Alt-F9
01	Alt-Esc	3E	F4	71	Alt-F10
02	Alt-Leertaste	3F	F5	72	Strg-Druck
04	Strg-Einfg	40	F6	73	Strg-←
05	Shift-Einfg	41	F7	74	Strg-→
06	Strg-Entf	42	F8	75	Strg-Ende
07	Shift-Entf	43	F9	76	Strg-Bild↓
08	Alt-Rückschritt	44	F10	77	Strg-Pos1
09	Alt-Shift-Rückschritt	47	Pos1	78	Alt-1
0F	Shift-Tab	48	↑	79	Alt-2
10	Alt-Q	49	Bild↑	7A	Alt-3
11	Alt-W	4B	←	7B	Alt-4
12	Alt-E	4C	Zahlenblock 5 (ohne Num)	7C	Alt-5
13	Alt-R	4D	→	7D	Alt-6
14	Alt-T	4E	Alt-GrauPlus	7E	Alt-7
15	Alt-Z (engl.: Alt-Y)	4F	Ende	7F	Alt-8
16	Alt-U	50	↓	80	Alt-9
17	Alt-I	51	Bild↓	81	Alt-0
18	Alt-O	52	Einfg	82	Alt-GrauMinus
19	Alt-P	53	Entf	83	Alt-Zahlenblock-Enter
1A	Alt-Ü (engl.: Alt-LftBrack)	54	Shift-F1	84	Strg-Bild↑
1B	Alt-+ (engl.: Alt-RgtBrack)	55	Shift-F2	85	F11
1E	Alt-A	56	Shift-F3	86	F12
1F	Alt-S	57	Shift-F4	87	Shift-F11
20	Alt-D	58	Shift-F5	88	Shift-F12
21	Alt-F	59	Shift-F6	89	Strg-F11
22	Alt-G	5A	Shift-F7	8A	Strg-F12
23	Alt-H	5B	Shift-F8	8B	Alt-F11
24	Alt-J	5C	Shift-F9	8C	Alt-F12
25	Alt-K	5D	Shift-F10	8D	Strg-↑
26	Alt-L	5E	Strg-F1	8E	Strg-GrauMinus
27	Alt-Ö (engl.: Alt-SemiCol)	5F	Strg-F2	8F	Strg-5 (ohne Num)
28	Alt-Ä (engl. :Alt-Quote)	60	Strg-F3	90	Strg-GrauPlus

Tabelle O4.7: Tasten-Scancodes (alle Codes in hexadez. Schreibweise), Teil1 von 2

Code	Taste	Code	Taste	Code	Taste
29	Alt-# (engl.:Alt-OpQuote)	61	Strg-F4	91	Strg-↓
2B	Alt-< (engl.: Alt-BkSlash)	62	Strg-F5	94	Strg-Tab
2C	Alt-Y (engl: Alt-Z)	63	Strg-F6	97	Alt-Pos1
2D	Alt-X	64	Strg-F7	98	Alt-↑
2E	Alt-C	65	Strg-F8	99	Alt-Bild↑
2F	Alt-V	66	Strg-F9	9B	Alt-←
30	Alt-B	67	Strg-F10	9D	Alt-→
31	Alt-N	68	Alt-F1	9F	Alt-Ende
32	Alt-M	69	Alt-F2	A0	Alt-↓
33	Alt-Komma	6A	Alt-F3	A1	Alt-Bild↓
34	Alt-Punkt	6B	Alt-F4	A2	Alt-Einfg
35	Alt-Minus (engl.: Alt-Slash)	6C	Alt-F5	A3	Alt-Entf
37	Alt-GrauerStern	6D	Alt-F6	A5	Alt-Tab
3B	F1	6E	Alt-F7		
3C	F2	6F	Alt-F8		

Tabelle 04.7: Tasten-Scancodes (alle Codes in hexadezimaler Schreibweise), Teil 2 von 2

Taste	Code	Shift+Taste	Strg+Taste	Alt+Taste
Keine Taste (NoKey)	00			
F1	3B	54	5E	68
F2	3C	55	5F	69
F3	3D	56	60	6A
F4	3E	57	61	6B
F5	3F	58	62	6C
F6	40	59	63	6D
F7	41	5A	64	6E
F8	42	5A	65	6F
F9	43	5B	66	70
F10	44	5C	67	71
F11	85	87	89	8B
F12	86	88	8A	8C
Pos1	47	77	97	
↑	48	8D	98	
Bild↑	49	84	99	
←	4B	73	9B	
Mittlere Ziffernblocktaste	4C	8F		
→	4D	74	9D	
Ende	4F	75	9F	
↓	50	91	A0	
Bild↓	51	76	A1	
Einfg	52	05	04	A2
Entf	53	07	06	A3
Tab	8	0F	94	A5
GrauPlus	90	4E		

Tabelle 04.8: Sondertasten-Scancodes (alle Codes in hexadezimaler Schreibweise)

Eine Liste der Scancodes für besondere Tasten und Kombinationen mit [Shift], [Alt] und [Strg] zeigt die Schnellreferenz in Tabelle O4.8.

4.22.3 Konstanten, Typen und Variable

Konstanten

AltPrefix: Byte = 0	Alternativer Index für die Alt-Taste. Nur unter Unix.
CtrlPrefix: Byte = 0	Alternativer Index für die Strg-Taste. Nur unter Unix.
ShiftPrefix: Byte = 0	Alternativer Index für die Shift-Taste. Nur unter Unix.

Fehlercodes:

errKbdBase = 1010	Basiswert für die Fehlerkonstanten der Tastaturroutinen.
errKbdInitError = errKbdBase + 0	Tastaturtreiber konnte nicht initialisiert werden.
errKbdNotImplemented = errKbdBase + 1	Tastaturtreiber ist nicht implementiert.

Die Konstanten für die Modifizierertasten (für *ShiftState*):

Konstante	Code	Taste
kbLeftShift	1	Modifizierer der linken Shifttaste.
kbRightShift	2	Modifizierer der rechten Shifttaste.
kbShift	kbLeftShift **or** kbRightShift	Modifizierer der Shifttaste.
kbCtrl	4	Modifizierer der Steuerungstaste (Strg).
kbAlt	8	Modifizierer der Alt-Taste.

Die Konstanten für die Tasten des Cursor- und Ziffernblocks:

Konstante	Code	Taste
kbdHome	$FF20	Pos1
kbdUp	$FF21	↑
kbdPgUp	$FF22	Bild↑
kbdLeft	$FF23	←
kbdMiddle	$FF24	Mittlere Taste des Zahlenblocks (Zahl 5).
kbdRight	$FF25	→
kbdEnd	$FF26	Ende
kbdDown	$FF27	↓
kbdPgDn	$FF28	Bild↓
kbdInsert	$FF29	Einfg
kbdDelete	$FF2A	Entf

Die Konstanten für das Drücken der Funktionstasten:

Konstante	Code	Taste
kbdF1	$FF01	F1 wurde gedrückt.
kbdF2	$FF02	F2 wurde gedrückt.
kbdF3	$FF03	F3 wurde gedrückt.
kbdF4	$FF04	F4 wurde gedrückt.
kbdF5	$FF05	F5 wurde gedrückt.
kbdF6	$FF06	F6 wurde gedrückt.
kbdF7	$FF07	F7 wurde gedrückt.
kbdF8	$FF08	F8 wurde gedrückt.
kbdF9	$FF09	F9 wurde gedrückt.
kbdF10	$FF0A	F10 wurde gedrückt.
kbdF11	$FF0B	F12 wurde gedrückt.
kbdF12	$FF0C	F12 wurde gedrückt.
kbdF13	$FF0D	F13 wurde gedrückt.
kbdF14	$FF0E	F14 wurde gedrückt.
kbdF15	$FF0F	F15 wurde gedrückt.
kbdF16	$FF10	F16 wurde gedrückt.
kbdF17	$FF11	F17 wurde gedrückt.
kbdF18	$FF12	F18 wurde gedrückt.
kbdF19	$FF13	F19 wurde gedrückt.
kbdF20	$FF14	F20 wurde gedrückt.

Die Windows-Tasten:

Konstante	Code	Taste
kbdLWin	$FF15	Linke Windows-Taste wurde gedrückt.
kbdRWin	$FF16	Rechte Windows-Taste wurde gedrückt.
kbdApps	$FF17	Die Anwendungstaste (Popup-Menü) wurde gedrückt.

Kennungen für die Flags von *TKeyCode*:

Konstante	Wert	Beschreibung
kbASCII	$00	Events für ASCII-Code.
kbUniCode	$01	Events für Unicode.
kbFnKey	$02	Funktionstaste wurde gedrückt.
kbPhys	$03	Physikalisches Tastenereignis.
kbReleased	$04	Event beim Loslassen der Taste

Strings für die Tastenbeschreibungen in den Routinen zur Beschreibung der Tastatur-Events:

SAnd: *String* = 'AND'	Diese Konstante wird den Routinen, die die Tastaturevents beschreiben, als Wort »And« in Tastenbeschreibungen verwendet. Der String kann bei Bedarf für die Lokalisierung der Beschreibungen geändert werden.

SKeyPad: Array[0..($FF2F - kbdHome)] *of String* = ('Home', 'Up', 'PgUkp', 'Left', 'Middle', 'Right', 'End', 'Down', 'PgDn', 'Insert', 'Delete', '', '', '', '', '')	Die Konstante beschreibt alle Tasten des Ziffern-/Cursorblocks. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.
SLeftRight: *array*[1..2] *of String* = ('LEFT', 'RIGHT')	Die Konstante beschreibt linke und rechte Tasten. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.
SScanCode: *String* = 'Key with scancode '	Die Konstante enthält einen String für den Hinweis auf Scancode-Events. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.
SShift: Array[1..3] *of String* = ('SHIFT', 'CTRL', 'ALT')	Diese Konstante enthält die Beschreibungen der Modifizierertasten. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.
SUnicodeChar: *String* = 'Unicode character '	Die Konstante enthält einen String für den Hinweis auf Unicode-Events. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.
SUnknownFunctionKey: *String* = 'Unknown function key : '	Die Konstante enthält einen String mit dem Hinweis, daß eine unbekannte Funktionstaste betätigt wurde. Sie wird von den Routinen zur Beschreibung der Tastenevents benötigt und kann bei Bedarf lokalisiert werden.

Typdeklarationen

```
TKeyboardDriver = record
  InitDriver              : procedure;
  DoneDriver              : procedure;
  GetKeyEvent             : function: TKeyEvent;
  PollKeyEvent            : function: TKeyEvent;
  GetShiftState           : function: Byte;
  TranslateKeyEvent       : function(KeyEvent: TKeyEvent): TKeyEvent;
  TranslateKeyEventUniCode: function(KeyEvent: TKeyEvent): TKeyEvent;
end;
```

Mit dem Record *TKeyboardDriver* (siehe auch Seite O100) kann ein eigener Tastaturtreiber mit der Funktion *SetKeyboardDriver* (siehe Seite O115) installiert werden. Die verschiedenen Felder des Records korrespondieren mit den unterschiedlichen Funktionen des Interfaces der Unit *Keyboard*. Weitere Informationen dazu können auf Seite O100 nachgelesen werden.

```
TKeyEvent = Cardinal;
```

Der Datentyp *TKeyEvent* ist der Grundtyp für alle Tastaturevents. Der Tastenanschlag wird in die 4 Byte des Typs *TKeyEvent* kodiert. Die verschiedenen Felder des Tastendrucks können mit einem Typecast des *TKeyEvents* auf den Typ *TKeyRecord* ermittelt werden.

```
TKeyRecord = packed record
  KeyCode   : Word;
  ShiftState: Byte;
  Flags     : Byte;
end;
```

Der *TKeyRecord* ist die Umsetzung des *TKeyEvents*. Die Bedeutung der einzelnen Felder ist in Tabelle O4.9 erläutert.

Feld	Bedeutung
KeyCode	Abhängig von den Flags entweder die physikalische Darstellung einer Taste (unter DOS der Scancode, ein Paar von ASCII-Codes) oder das übersetzte ASCII/Unicode-Zeichen.
ShiftState	Der Shiftstate zum Zeitpunkt, als die Taste gedrückt wurde (oder kurz danach).
Flags	Die Definition, wie *KeyCode* zu interpretieren ist.

Tabelle O4.9: Die Struktur des Records TKeyRecord

Der *ShiftState* wird mit den verschiedenen dafür vorgesehenen Konstanten geprüft und die Flags im letzten Byte mit den Konstanten *kbASCII*, *kbUniCode*, *kbFnKey*, *kbPhys* und *kbReleased*.
Liefern zwei Tasten den selben Zeichencode zurück, gibt es keine Möglichkeit, festzustellen, welche gedrückt wurde (zum Beispiel die normale oder graue [+]-Taste). Wenn man solche Unterschiede herausfinden möchte, muß mit den unübersetzten Tastencodes gearbeitet werden, die jedoch systemabhängig sind. Für diese Aufgabe können systemabhängige Konstanten definiert werden, die möglicherweise den selben Namen oder unterschiedliche Werte besitzen.

4.22.4 Prozeduren und Funktionen

DoneKeyboard

```
procedure DoneKeyboard;
```

Wenn der Tastaturtreiber aktiv ist, entfernt *DoneKeyboard* die Tastaturschnittstelle. Ist sie nicht initialisiert, macht die Prozedur nichts. Der Aufruf führt dazu, daß jeglicher belegter Speicher aufgeräumt und die Konsole oder das Terminal in seinen Ursprungszustand versetzt wird – also in den Status vor dem Aufruf von *InitKeyboard*. Diese Prozedur sollte beim Verlassen des Programms aufgerufen werden. Erfolgt der Aufruf nicht oder schlägt fehl, wird das Terminal oder die Konsole in einem Zustand belassen, in dem sie nicht mehr brauchbar ist. Wie das genau aussieht, hängt von der Plattform ab, unter der das Programm läuft.
Unter Unix restauriert der voreingestellte Tastaturtreiber die Zeilenenden außerdem von *System.Output* nach #10.
Ein Beispiel für den Aufruf dieser Prozedur wird bei den meisten anderen Funktionen gezeigt.
Siehe auch: *InitKeyBoard*.

FunctionKeyName

```
function FunctionKeyName(KeyCode: Word) : String
```

FunctionKeyName gibt einen String mit dem Namen der Funktionstaste für den Code *KeyCode* zurück. Dabei kann es sich um eine [Fxx]-Taste oder eine der Cursortasten handeln.
Fehler: In Fällen, in denen *KeyCode* keinen Code einer Funktionstaste enthält, wird der String *SUnknownFunctionKey* mit dem angehängten KeyCode zurückgegeben.
Siehe auch: *ShiftStateToString* und *KeyEventToString*.

```
program Example8; (* Beispiel für die Funktion FunctionKeyName *)
uses
  Keyboard;
var
  K: TKeyEvent;
```

```
begin
  InitKeyboard;
  WriteLn('Drücken Sie eine Funktionstaste oder "q", um zu beenden.');
  repeat
    K := GetKeyEvent;
    K := TranslateKeyEvent(K);
    if IsFunctionKey(k) then begin
      Write('Funktionstaste erhalten: ');
      WriteLn(FunctionKeyName(TKeyRecord(K).KeyCode));
    end;
  until GetKeyEventChar(K) = 'q';
  DoneKeyboard;
end.
```

Listing 04.50: kbdex/ex8.pp

GetKeyboardDriver

```
procedure GetKeyboardDriver(var Driver: TKeyboardDriver);
```

GetKeyBoardDriver gibt in *Driver* den aktuell gültigen Tastaturtreiber zurück. Mit dieser Funktion kann ein vorhandener Tastaturtreiber (siehe Seite O100) erweitert werden.
Mehr Informationen zum Lesen und Setzen des Tastaturtreibers können ab Seite O100 nachgelesen werden.
Siehe auch: *SetKeyboardDriver*.

GetKeyEvent

```
function GetKeyEvent: TKeyEvent;
```

GetKeyEvent gibt das letzte Tastatur-Ereignis zurück oder wartet auf einen Event, wenn gerade kein Ereignis anliegt. Eine nicht-blockierende Version dieses Aufrufs ist mit *PollKeyEvent* verfügbar.
Die zurückgegebene Taste wird als *TKeyEvent*-Variable kodiert und ist normalerweise der Scancode der physikalischen Taste (der Scancode ist treiberabhängig). Er kann mit den Funktionen *TranslateKeyEvent* oder *TranslateKeyEventUniCode* übersetzt werden. Im Abschnitt zu den Typdeklarationen ab Seite O107 ist beschrieben, wie die Tasten definiert sind.
Fehler: Falls keine Taste gelesen werden konnte, beispielsweise weil sie der Treiber nicht unterstützt, wird der Wert 0 zurückgegeben.
Siehe auch: *PutKeyEvent, PollKeyEvent, TranslateKeyEvent* und *TranslateKeyEventUniCode*.

```
program example1; (* Beispiel für die Funktion GetKeyEvent *)
uses
  keyboard;
var
  K: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Drücken Sie eine Taste, "q" beendet das Beispiel.');
  repeat
    K := GetKeyEvent;
    K := TranslateKeyEvent(K);
    Write('Tastaturevent erhalten mit ');
    case GetKeyEventFlags(K) of
      kbASCII   : WriteLn('ASCII-Taste');
      kbUniCode : WriteLn('Unicode-Taste');
      kbFnKey   : WriteLn('Funktionstaste');
```

```
      kbPhys    : WriteLn('Physikalischer Taste');
      kbReleased: WriteLn('Freigegebenem Tastaturevent');
    end;
    WriteLn('Erhaltene Taste: ', KeyEventToString(K));
  until GetKeyEventChar(K) = 'q';
  DoneKeyBoard;
end.
```

Listing 04.51: kbdex/ex1.pp

GetKeyEventChar

```
function GetKeyEventChar(KeyEvent: TKeyEvent): Char;
```

GetKeyEventChar liefert den Anteil des Tastencodes des angegebenen Records *KeyEvent,* falls dieser einen übersetzten Zeichencode in *Keycode* enthält. Der Zeichencode ist einfach nur der ASCII-Code der gedrückten Zeichentaste.
Die Funktion ist nur bei Funktionstasten sinnvoll, nicht bei alphanumerischen Tasten.
Ein Beispiel für die Funktion ist bei *GetKeyEvent* abgebildet.
Siehe auch: *GetKeyEventUniCode, GetKeyEventShiftState, GetKeyEventFlags, GetKeyEventCode* und *GetKeyEvent*.

GetKeyEventCode

```
function GetKeyEventCode(KeyEvent: TKeyEvent): Word;
```

GetKeyEventCode gibt den übersetzten Funktionstastenanteil des übergebenen Records *KeyEvents* zurück, falls dieser Record eine übersetzte Funktionstaste enthält. War die gedrückte Taste keine Funktionstaste oder Taste des Cursorblocks, wird ein ASCII-Null zurückgemeldet.
Siehe auch: *GetKeyEventUniCode, GetKeyEventShiftState, GetKeyEventFlags, GetKeyEventChar* und *GetKeyEvent*.

```
program Example2; (* Beispiel für die Funktion GetKeyEventCode *)
uses
  keyboard;
var
  K: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Drücken Sie eine erweiterte Taste, "q" beendet das Programm.');
  repeat
    K := GetKeyEvent;
    K := TranslateKeyEvent(K);
    if GetKeyEventFlags(K) <> KbfnKey then
      WriteLn('Keine erweiterte Taste')
    else begin
      Write('Erhaltene Taste (', GetKeyEventCode(K));
      WriteLn(') : ', KeyEventToString(K));
    end;
  until GetKeyEventChar(K) = 'q';
  DoneKeyboard;
end.
```

Listing 04.52: kbdex/ex2.pp

GetKeyEventFlags

```
function GetKeyEventFlags(KeyEvent: TKeyEvent): Byte;
```

GetKeyEventFlags gibt den Flags-Anteil des übergebenen *TKeyEvent*-Records zurück.
Ein Beispiel für diese Funktion ist bei *GetKeyEvent* gezeigt.

Siehe auch: *GetKeyEventUniCode, GetKeyEventShiftState, GetKeyEventCode, GetKeyEventChar* und *GetKeyEvent*.

GetKeyEventShiftState

```
function GetKeyEventShiftState(KeyEvent: TKeyEvent): Byte;
```

GetKeyEventShiftState gibt den Shiftstatus des angegebenen *TKeyEvent*-Records *KeyEvent* zurück. Hiermit wird ermittelt, ob eine der Modifizierer-Tasten [Shift], [Alt] oder [Strg] zusammen mit der normalen Taste gedrückt wurde. Wurde keine der Sondertasten betätigt, wird 0 zurückgegeben.
Diese Funktion zeitigt nicht immer das erwartete Ergebnis. In einem X-Terminal unter Unix ist nicht garantiert, daß die Modifizierertasten immer funktionieren.
Siehe auch: *GetKeyEventUniCode*, *GetKeyEventFlags*, *GetKeyEventCode*, *GetKeyEventChar* und *GetKeyEvent*.

```
program Example3;
(* Beispiel für die Funktion GetKeyEventShiftState *)
uses
  keyboard;
var
  K: TKeyEvent;
  S: Byte;
begin
  InitKeyBoard;
  Write('Drücken Sie eine Taste in Kombination mit Strg/Shift/Alt');
  WriteLn(' oder beenden Sie mit "q".');
  repeat
    K := GetKeyEvent;
    K := TranslateKeyEvent(K);
    S := GetKeyEventShiftState(K);
    if S = 0 then
      WriteLn('Keine Sondertaste gedrückt')
    else begin
      WriteLn('Ermittelte Sondertasten: ', ShiftStateToString(K, False));
      WriteLn('Empfangene Taste: ', KeyEventToString(K));
    end;
  until GetKeyEventChar(K) = 'q';
  DoneKeyboard;
end.
```

Listing 04.53: kbdex/ex3.pp

GetKeyEventUniCode

```
function GetKeyEventUniCode(KeyEvent: TKeyEvent): Word;
```

GetKeyEventUniCode liefert den Unicode-Anteil des angegebenen *TKeyEvent*-Records *KeyEvent*, falls dieser ein übersetztes Unicode-Zeichen enthält.
Siehe auch: *GetKeyEventShiftState, GetKeyEventFlags, GetKeyEventCode, GetKeyEventChar* und *GetKeyEvent.*

InitKeyboard

```
procedure InitKeyboard;
```

InitKeyboard initialisiert den Tastaturtreiber. Ist er schon aktiv, kehrt die Prozedur sofort zurück.
Ist der Treiber initialisiert, kümmert er sich um alles, was nötig ist, daß die Tastatur richtig ausgewertet werden kann, wozu auch das Anfordern von Speicher, das Initialisieren des Terminals und so weiter gehört.

Diese Prozedur muß einmal aufgerufen werden, bevor andere Tastaturfunktionen benutzt werden. Beim Beenden des Programms muß der Treiber mit *DoneKeyboard* wieder abgeschaltet werden oder es wird zur Laufzeit mit dem Aufruf von *SetKeyboardDriver* ein anderer Treiber installiert.
Unter Unix setzt der voreingestellte Tastaturtreiber das Terminal in den Raw-Modus. In diesem Modus wechselt der aktuelle Zeilenumbruch zum Zeilenvorschub ohne Wagenrücklauf. Das heißt, daß der Cursor zwar eine Zeile nach unten wandert, dabei sich aber die X-Koordinate nicht ändert, was dem Verhalten unter DOS/Windows entspricht. Um dieses Verhalten zu kompensieren, setzt der Treiber das Zeilenende auf *System.Output*, was dem unter DOS, Windows und OS/2 geläufigen #13#10 entspricht.
Siehe auch: *DoneKeyboard* und *SetKeyboardDriver*.
Ein Beispiel ist bei den meisten anderen Funktionen abgebildet.

IsFunctionKey

```
function IsFunctionKey(KeyEvent: TKeyEvent): Boolean;
```

IsFunctionKey ergibt *True,* wenn die in *KeyEvent* angegebene Taste eine Funktionstaste war.
Siehe auch: *GetKeyEvent.*

```
program example1;    (* Beispiel für die Funktion GetKeyEvent *)
uses
  keyboard;
var
  K: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Press keys, press "q" to end.');
  repeat
    K := GetKeyEvent;
    K := TranslateKeyEvent(K);
    if IsFunctionKey(K) then
      WriteLn('Got function key : ', KeyEventToString(K))
    else
      WriteLn('not a function key.');
  until GetKeyEventChar(K) = 'q';
  DoneKeyBoard;
end.
```

Listing 04.54: kbdex/ex7.pp

KeyEventToString

```
function KeyEventToString(KeyEvent: TKeyEvent): String;
```

KeyEventToString übersetzt das Tastaturereignis in *KeyEvent* in die lesbare Beschreibung der gedrückten Taste, dazu werden die Konstanten im Konstantenabschnitt ausgegeben.
Fehler: Wird eine unbekannte Taste gedrückt, wird der Scancode mit vorangestelltem Konstantenstring *SScanCode* zurückgegeben.
Siehe auch: *FunctionKeyName, ShiftStateToString*.
Ein Beispiel ist bei den meisten anderen Funktionen abgebildet.

KeyPressed

```
function KeyPressed: Boolean;
```

KeyPressed prüft die Tastatur-Eventqueue darauf, ob ein Tastenevent anliegt und ergibt *True*, falls das der Fall ist. Die Funktion ruft dazu nur *PollKeyEvent* auf und prüft auf ein gültiges Ergebnis.

Siehe auch: *PollKeyEvent, GetKeyEvent*.

PollKeyEvent

```
function PollKeyEvent: TKeyEvent;
```

PollKeyEvent prüft, ob ein Tastaturereignis anliegt, und gibt es zurück, wenn eines gefunden wird. Liegt kein Ereignis an, gibt die Funktion den Wert 0 zurück.
Es muß beachtet werden, daß die Funktion den Tastendruck nicht aus der Liste nimmt, er steht also in der Eventliste immer noch für die Funktion *GetKeyEvent* zur Verfügung.
Siehe auch: *PutKeyEvent, GetKeyEvent*.

```
program example4; (* Beispiel für die Funktion PollKeyEvent *)
uses
  keyboard;
var
  K: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Drücken Sie eine Taste, "q" beendet das Programm.');
  repeat
    K := PollKeyEvent;
    if k <> 0 then begin
      K := GetKeyEvent;
      K := TranslateKeyEvent(K);
      WriteLn;
      WriteLn('Taste erhalten: ', KeyEventToString(K));
    end else
      Write('.');
  until GetKeyEventChar(K) = 'q';
  DoneKeyBoard;
end.
```

Listing 04.55: kbdex/ex4.pp

PollShiftStateEvent

```
function PollShiftStateEvent: TKeyEvent;
```

PollShiftStateEvent gibt den aktuellen Shiftstatus in einem *TKeyEvent* zurück. Das Ergebnis ist 0, wenn kein Tastaturereignis anliegt.
Siehe auch: *PollKeyEvent, GetKeyEvent*.

```
program example6; (* Beispiel für die Funktion PollShiftStateEvent *)
uses
  keyboard;
var
  K: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Drücken Sie eine Taste, "q" beendet das Programm.');
  repeat
    K := PollKeyEvent;
    if k <> 0 then begin
     K := PollShiftStateEvent;
      WriteLn('Sondertaste erhalten: ', ShiftStateToString(K, False));
      // Consume the key.
      K := GetKeyEvent;
      K := TranslateKeyEvent(K);
    end else
```

```
      Write('.');
  until (GetKeyEventChar(K) = 'q');
  DoneKeyBoard;
end.
```

Listing 04.56: kbdex/ex6.pp

PUTKEYEVENT

```
procedure PutKeyEvent(KeyEvent: TKeyEvent)
```

PutKeyEvent fügt den angegebenen *KeyEvent* in die Eingabewarteschlange ein. Es muß beachtet werden, daß abhängig von der aktuellen Implementation diese Warteschlange immer nur einen Wert aufnehmen kann, das heißt, daß wenn *PutKeyEvent* mehrfach aufgerufen wird, immer nur die letzte gedrückte Taste erhalten bleibt.
Siehe auch: *PollKeyEvent, GetKeyEvent*.

```
program example5; (* Beispiel für die Prozedur PutKeyEvent *)
uses
  keyboard;
var
  K, K2: TKeyEvent;
begin
  InitKeyBoard;
  WriteLn('Drücken Sie eine Taste, "q" beendet das Programm.');
  K2 := 0;
  repeat
    K := GetKeyEvent;
    if k <> 0 then begin
     if k2 mod 2 = 0 then K2 := K + 1 else K2 := 0;
      K := TranslateKeyEvent(K);
      WriteLn('Taste erhalten: ', KeyEventToString(K));
      if K2 <> 0 then begin
        PutKeyEvent(k2);
        K2 := TranslateKeyEVent(K2);
        WriteLn('Taste gesandt: ', KeyEventToString(K2))
      end
    end
  until GetKeyEventChar(K) = 'q';
  DoneKeyBoard;
end.
```

Listing 04.57: kbdex/ex5.pp

RAWREADKEY

```
function RawReadKey: Char;
```

-- keine Beschreibung verfügbar --

RAWREADSTRING

```
function RawReadString : String;
```

-- keine Beschreibung verfügbar --

RESTORESTARTMODE

```
procedure RestoreStartMode;
```

-- keine Beschreibung verfügbar --

SetKeyboardDriver

```
function SetKeyboardDriver(const Driver: TKeyboardDriver): Boolean;
```

SetKeyBoardDriver setzt dem Tastaturtreiber auf *Driver*, falls der aktuelle Treiber noch nicht initialisiert ist. Ist der aktuelle Tastaturtreiber bereits aktiv, macht *SetKeyboard-Driver* nichts mehr. Vor dem Setzen des Treibers sollte der aktuelle aktive Treiber mit dem Aufruf *DoneKeyboard* abgeschaltet werden.
Die Funktion meldet *True*, wenn der Treiber gesetzt werden konnte, *False*, wenn nicht.
Weitere Informationen zum Setzen des Tastaturtreibers sind im Abschnitt ab Seite O100 nachzulesen.
Siehe auch: *GetKeyboardDriver, DoneKeyboard.*

ShiftStateToString

```
function ShiftStateToString(KeyEvent: TKeyEvent;
                            UseLeftRight: Boolean): String;
```

ShiftStateToString liefert einen String mit der Beschreibung des Shiftstatus der Taste in *KeyEvent* zurück, was auch ein leerer String sein kann. Der Shiftstatus wird mit den Strings in der Konstante *SShift* beschrieben.
Ein Beispiel ist bei *PollShiftStateEvent* gezeigt.
Siehe auch: *FunctionKeyName, KeyEventToString*.

TranslateKeyEvent

```
function TranslateKeyEvent(KeyEvent: TKeyEvent): TKeyEvent;
```

TranslateKeyEvent führt die ASCII-Übersetzung von *KeyEvent* durch. Ein physikalischer Tastendruck wird in eine Funktionstastendefinition umgesetzt, falls eine Funktionstaste gedrückt wurde, und die physikalische Taste in das ASCII-Zeichen, wenn es sich um eine normale Taste handelte.
Ein Beispiel für die Funktion ist bei *GetKeyEvent* gezeigt.
Siehe auch: *TranslateKeyEventUniCode*.

TranslateKeyEventUniCode

```
function TranslateKeyEventUniCode(KeyEvent: TKeyEvent): TKeyEvent;
```

TranslateKeyEventUniCode führt die Unicode-Übersetzung von *KeyEvent* durch. Diese Funktion ist noch nicht auf allen Plattformen implementiert.
Fehler: Ist die Funktion auf der gewählten Zielplattform nicht implementiert, wird der *ErrorCode* der Unit *System* auf *errKbdNotImplemented* gesetzt.
Siehe auch: *TranslateKeyEvent*.

Stichwortverzeichnis (Online-Daten)

A

AnsiStrings ... 24
Archiv-Attribut ... 8
ASCII-#0-Zeichen ... 26
Attribut für alle Dateien ... 8

B

Betriebssystemaufruf ... 20
Bildschirm aktualisieren ... 75
Bildschirm löschen ... 77, 85
Bildschirm, Array f. physikalischen ... 84
Bildschirm, schreiben in ... 75
Bildschirm, Zeichen auf ... 83
Bildschirm-Aktualisierung ... 89
Bildschirmmodus ... 77, 88
Bildschirmposition, an die Daten geschrieben werden ... 84
Bildschirmpuffergröße ... 84
Bildschirmtreiber abschalten ... 85
Bildschirm-Vorder-/Hintergrundfarbe ... 75
Bildschirm-Zugriff ... 75
Blinkbit ... 75
Botschafts-Warteschlange ... 60

C

Codepage ... 81
CPU-Flags ... 9
Cursor-/Ziffernblock-Tasten ... 105
Cursortyp ... 86

D

Datei, versteckte ... 8
Dateiänderung, Uhrzeit der ... 18
Dateiattribute ... 17
Dateiattribute-Konstanten ... 8
Dateiaufrufe ... 8
Dateimodus-Konstanten ... 8
Dateiname zerlegen ... 16
Dateinamen-Längen ... 9
Dateisuche ... 9, 15
Dateisystem ... 8
Datum ... 8
DOS ... 8

F

Freie Byte auf Laufwerk ... 11
Funktionstasten ... 106

I

Int64 ... 24
Interprozeßkommunikation ... 58
Interruptvektorentabelle ... 22

L

Laufwerkgröße ... 12
LongInt ... 24

M

Mausbewegungsevent ... 93
Mausposition ... 96
Maussteuerung ... 93
Maustreiber ... 95
Mauszeiger ... 98
Meldungswarteschlange ... 64
Messages ... 58
Modifizierer-Tasten ... 105

N

Nachrichten ... 58

P

Pfadangaben ... 14

R

Ressourcen-Freigabe ... 14

S

Schreibschutzattribut ... 8
Schriften ... 81
Semaphore ... 61
Semaphoren ... 58
Shared Memory ... 58
Shiftstatus ... 111
Shortstrings ... 24
Signalübermittlung ... 58
SizeInt ... 24
Socket ... 32
Softwareinterrupt ausführen ... 20
Sondertasten-Scancodes ... 104
Speicher, gemeinsam genutzter ... 58
String konvertieren ... 29
String kopieren ... 25
String, Speicher reservieren für ... 24
Stringvergleich ... 24
Systemdatei ... 8
Systemdatum ... 16
Systemuhr ... 22

T

Tastatur unter Unix ... 99
Tastaturereignis ... 99
Tastaturevent ... 107
Tastaturtreiber ... 99
Tastaturtreiber schreiben ... 100
Tastaturzugriffsschicht ... 99
Tastenevent übersetzen ... 115
Tasten-Scancodes ... 103
TCP/IP ... 33
Textbildschirm, Farben ... 80
Text-Eingabecursor, Aussehen ... 81

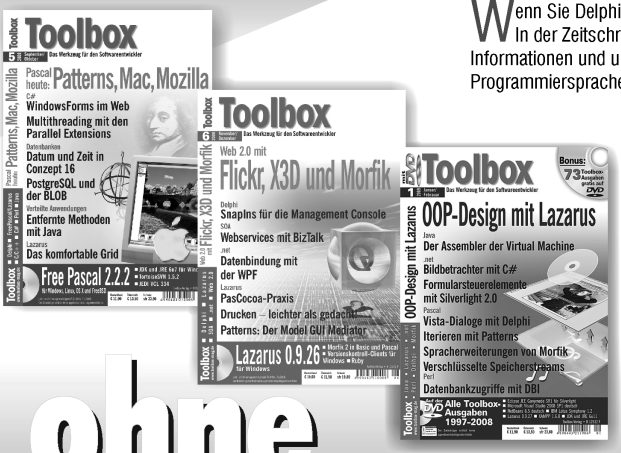
Uhrzeit ... 8
Umgebungsvariablen, Anzahl ... 13
Unicode-Zeichen, Tastaturevent ... 111
Unix-Zeitstempel ... 13
Unix-Zeitstempel konvertieren ... 23

Versionsnummer des Betriebssystems/Kernelversion ... 13
Verzeichnis-Attribut ... 8
Video-Subsystem initalisieren ... 89
Volume Label ... 8

W

Warteschleife ... 64
WideStrings ... 24
Wochentag ... 23

Programmieren



Wenn Sie Delphi-, C++-, Basic- oder Java-Entwickler sind: In der Zeitschrift Toolbox finden Sie wertvolle Tips, Insider-Informationen und unverzichtbares Hintergrundwissen zu Ihrer Programmiersprache. Und das seit mehr als 10 Jahren.

Jede Ausgabe der Toolbox bietet Ihnen Programmierlösungen, Tips und Tricks zur Systemprogrammierung, Anwendungsentwicklung und zum Komponentendesign unter Windows und Linux.

Die CD-ROM, die jeder Ausgabe beigefügt ist, ist eine direkte redaktionelle Erweiterung des Hefts. Hier finden Sie die Listings zu den Text-Beiträgen, zusätzlich Komponenten, Compiler, Routinen und Tools, SDKs und vieles mehr, speziell für Programmierer und oftmals exklusiv für Toolbox-Leser.

ohne Rumprobieren.

Profitieren auch Sie vom geballten Know-how praxiserfahrener Autoren und bestellen Sie das Schnupperabo mit drei Heften

zum Preis von zwei!

Per Fax an: 0 89 / 200 281 26
oder per Post an unten stehende Adresse.

Aktion: Mini-ABO

toolbox

An

Toolbox-Verlag
Toolbox
Abo-Service CSJ
Postfach 140220

80452 München

www.toolbox-mag.de

Ich bestelle 3 Ausgaben der Programmiererzeitschrift **Toolbox** im Mini-Abo zum Sonderpreis von nur 20,– € frei Haus. – Preis außerhalb Deutschland 23,– €.

Ich war in den letzten 12 Monaten kein Toolbox-Abonnent.

Name, Vorname

Straße, Nr.

PLZ, Ort

Datum, Unterschrift

Sollte sich meine Adresse ändern, erlaube ich der Deutschen Bundespost, meine neue Adresse dem Verlag mitzuteilen

Ich wünsche folgende Zahlungsweise (wie angekreuzt):

☐ Bequem und bargeldlos durch Bankabbuchung

Kontonummer | Bankleitzahl

Geldinstitut

☐ Durch Überweisung nach Erhalt der Rechnung.

PRESSE

Die Toolbox finden Sie im Bahnhofsbuchhandel und am Kiosk mit diesem Zeichen

Computerwissen für Praktiker C&L

Computer in der Box

Fabian Thorns (Hrsg.)

Das Virtualisierungs-Buch

Konzepte, Techniken und Lösungen

Die aktualisierte und deutlich erweiterte 2. Auflage des Bestsellers zu VMware, Xen, Parallels, Microsoft, Qemu, VirtualBox, VServer und weiteren Lösungen für Windows, Unix/Linux und Mac. Anwendungsübergreifend aufgebaut für die optimale Kombination von Produkten für jeden Einsatzbereich.

- **799 Seiten • Softcover • 2007**
- **EUR 49,90 (D) • ISBN 978-3936546-56-9**

Probekapitel und Inhaltsverzeichnis und die Möglichkeit zur Direktbestellung aller verfügbaren Bücher finden Sie auf www.cul.de

COUPON

Legen Sie diesen Coupon Ihrer Buchhandlung vor. Falls das gewünschte Buch gerade nicht vorrätig ist, kann sie es Ihnen innerhalb von 24 Stunden besorgen.

Hiermit bestelle ich folgende Bücher des C&L Computer- und Literaturverlags:

☐ Das Virtualisierungs-Buch • Konzepte, Techniken und Lösungen
ISBN 978-3936546-56-9

Dazu paßt:

☐ Wikis und Blogs • Planen, Einrichten, Verwalten
ISBN 978-3936546-44-4

Name: _____

Straße: _____

Ort: _____

Unser Gesamtprogramm finden Sie unter:

www.cul.de

Computer & Literatur Verlag

C&L-Bücher erhalten Sie im gut sortierten Buch- und Fachhandel oder über www.cul.de

Schon gelesen?

Testen Sie die freeX mit dem Miniabo!

- Kompetenz in Linux, BSD und Solaris
- Administration, Anwendung und Programmierung
- Systemsicherheit, Netzwerk und Datenbanken
- Hardware und Peripherie

Drei aktuelle Ausgaben freeX zum Preis von zwei: nur 23,80 Euro!

Ab jetzt immer mit DVD!

freeX ist eine Publikation des Computer- und Literaturverlag

www.cul.de

Per Fax an: 0 89 / 20 02 81-15
oder per Post an unten stehende Adresse,
oder formlos per E-Mail auf www.cul.de

Antwort

Computer & Literaturverlag
freeX Abo-Service CSJ
Postfach 140 220

D-80452 München

Alle 2 Monate neu im Bahnhofsbuchhandel und am Kiosk mit diesem Zeichen:

PRESSE

Ja, ich bestelle 3 Ausgaben der Zeitschrift **freeX** im Mini-Abo zum Sonderpreis von nur 23,80 € frei Haus.
Ich war in den letzten 12 Monaten kein freeX-Abonnent.

Name, Vorname: _____

Straße, Nr.: _____

PLZ, Ort: _____

Datum, Unterschrift: X _____

Sollte sich meine Adresse ändern, erlaube ich der Deutschen Bundespost, meine neue Adresse dem Verlag mitzuteilen.

Ich wünsche folgende Zahlungsweise (wie angekreuzt):

❑ Bequem und bargeldlos durch Bankabbuchung:

Kontonummer: _____ Bankleitzahl: _____

Geldinstitut: _____

❑ Durch Überweisung nach Erhalt der Rechnung.